#139 MAY

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

DR. DOBB'S JOURNAL

#138 MAY 1988

Avoiding Software PITFALLS

INSIDE IMAGE FILES

FASTER PATTERN MATCHING

LANGUAGES:

Assembler, C, Pascal

NEW COLUMN

Programming Paradigms

ox is the best

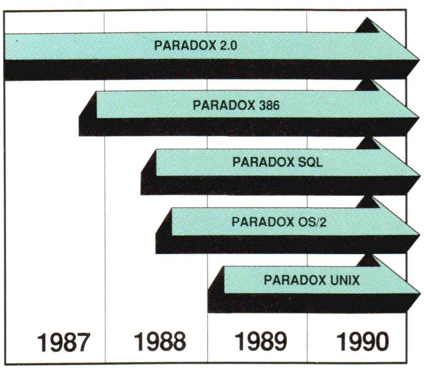
There's no power like Paradox Power

PROGRAM	Overall Power	Overall Performance	Overall Versatility	Single-Record Search (sec.)	Indexed Group Record Search (sec.)	Search without an Index (sec.)	Subtotal 100 Groups (sec.)	Subtotal on Two-File Join (sec.)	Three-File Join (sec.)	Subtotal on Three-File Join (sec.)	Four-File Join (sec.)	Many-to-Many Join (sec.)
Paradox	8.4	9.2	7.5	1.0	6	100	221	325	459	364	322	18
dBASE III PLUS	7.9	8.9	6.9	0.5	17	93	525	1013	646	449	445	39

*Source: Software Digest**

Paradox saves you from future shock

Trends for the future with Paradox



Paradox 386 allows users to take advantage of 16 Megabytes of Memory on a 386 machine. This allows Paradox users to work with databases that could in the past only be handled by minicomputers and mainframes.

Your investment today in Paradox applications is protected as new hardware and operating systems are used in your company. Paradox 2.0 applications will run unchanged on Paradox 386, Paradox OS/2, Paradox Unix and Paradox SQL! All versions of Paradox will be completely application and menu compatible. Paradox SQL will allow access to remote databases via SQL. Users will just type in a query as they normally would, and Paradox will translate that Query to SQL.

“Paradox 2.0 will do for the LAN what the spreadsheet did for the PC

David Schulman,
Bendix Aerospace ”

Paradox makes your network run like clockwork

Paradox is just as valuable to multi and network users as it is to single users. It runs smoothly, intelligently and so transparently that multiusers can access the same data at the same time—without either being aware of each other or getting in each other's way. It works exactly the same way whether you're flying solo or as part of the crew.

“Anyone who hasn't seen the network version of Paradox should take a look. Ansa has dramatically advanced the state of the art in multiuser network databases

Phil Lemmons,
BYTE

Paradox was a delight to use, both as a standalone product and from a local area network server.

Don Crabb,
InfoWorld ”

How to make your network network

To run Paradox 2.0 or the Paradox Network Pack on a network you need:

- Novell with Novell Advanced Netware version 2.0A or higher
- 3Com 3Plus with 3Com 3+ operating system version 1.0, 1.1 or higher
- IBM Token Ring or PC Network with IBM PC Local Area Network Program version 1.12 or higher
- Torus Tapestry version 1.4 or higher
- AT&T Starlan Network with AT&T PC 6300 Network Program version
- Other network configurations that are 100% compatible with DOS 3.1 and one of the listed networks

System Requirements for Single User:

- DOS 2.0 or higher
- IBM® PS/2 and PC, Compaq® PC families and other 100% compatibles
- 512K RAM
- Two disk drives, 3½-inch and 5¼-inch supported
- Compatible monochrome, color, or EGA monitor with adapter

System Requirements for the Network Workstation:

- DOS 3.1 or higher
- 640K RAM
- Any combination of hard, floppy, or no disk drives
- Compatible monochrome, color, or EGA monitor with adapter

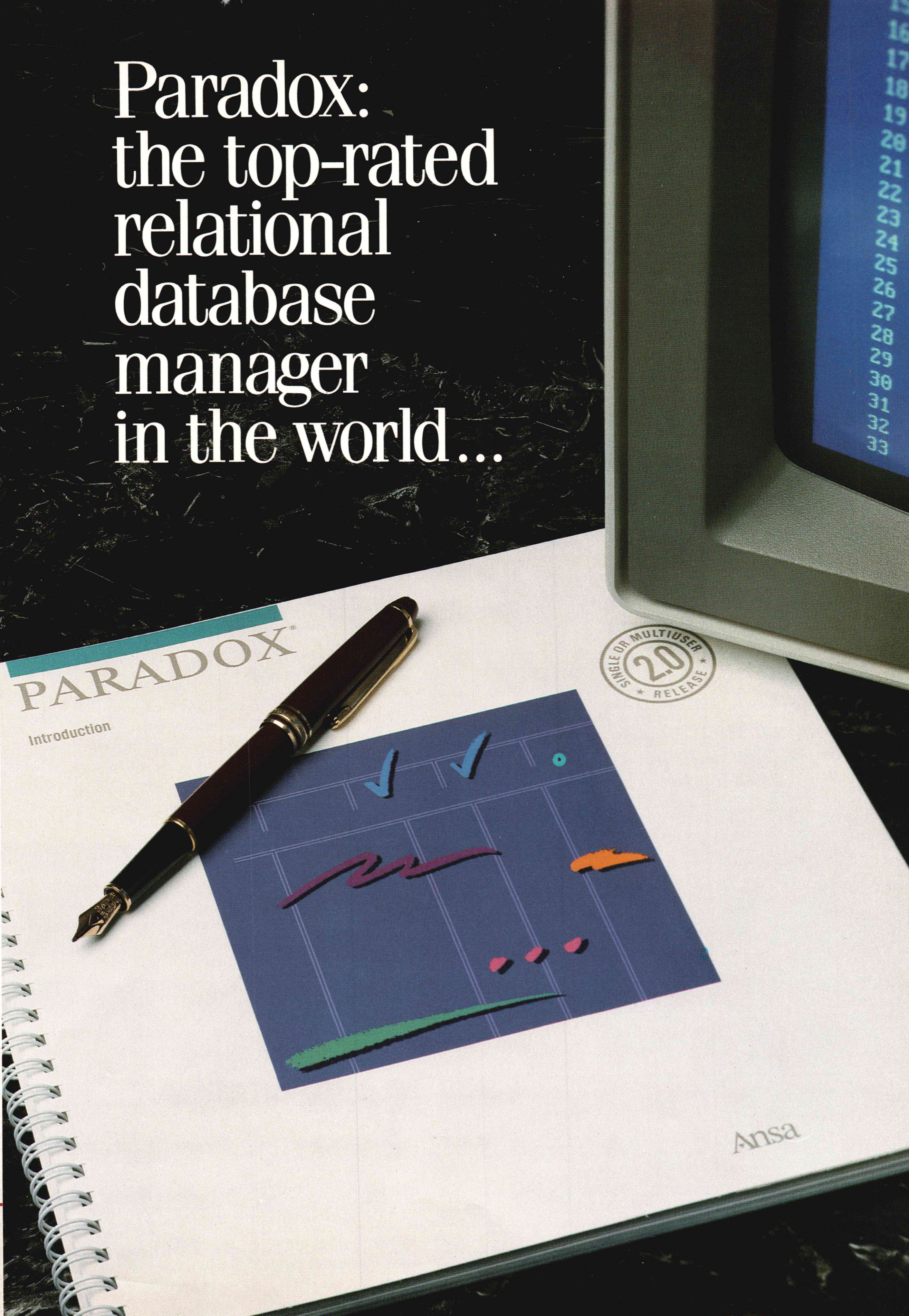
Optional Equipment:

- EMS and EEMS Boards: AST RAMpage Board,™ Intel Above Board® or other expanded memory adapters
- Printers: Compatible dot matrix, letter quality, or laser printer

**Reprinted with permission by Software Digest from its July 1987 report covering 12 relational database programs.*

†Test was designed and executed by NSTL. A 1,000-record and a 10,000-record file were joined. A short text field from the 1,000-record file and a numeric field from the 10,000-record file were selected (using the 1,000-record file indexes). The short text field was grouped and sorted in ascending order, the numeric field was subtotaled for each group, and the results output to a null printer. Test times from the last keystroke on the command sequence until return of program control were recorded and averaged.

DDJ 5-88


Paradox:
the top-rated
relational
database
manager
in the world...
PARADOX®
Introduction
SINGLE OR MULTIUSER
2.0
RELEASE
Ansa

Why Parado

"Paradox® is once again the top-rated program, with the latest version scoring even higher than last year's top score." (Software Digest's July 1987 Ratings Report—an independent comparative ratings report for selecting IBM PC Business software).

All tests for the Ratings Report were done by the prestigious National Software Testing Laboratory, Philadelphia, PA, and the message is crystal clear: there is no better relational database manager than Paradox.

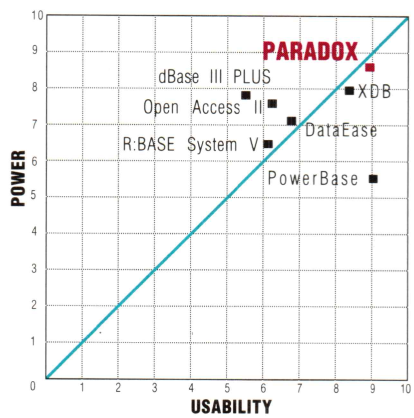
NSTL tested 12 different programs and amongst other results, discovered that Paradox is 3 times faster than dBASE; 6 times faster than R:BASE on a two-file join with subtotals test†.

Paradox does the impossible: combines ease-of-use with power and sophistication

Even if you're a beginner, Paradox is the only relational database manager that you can take out of the box and begin using right away.

Because Paradox employs state-of-the-art artificial intelligence technology, it does almost everything for you—except take itself out of the box.

If you've ever used 1-2-3® or dBASE®, you already know how to use Paradox. It has Lotus-like menus, and Paradox documentation includes "A Quick Guide to Paradox for Lotus users," and "A Quick Guide to Paradox for dBASE users."



*Source: Software Digest**

Ideal programs have high levels of both power and usability. Programs plotted in the upper righthand portion of the diagram above come closest to achieving that ideal.

> ❝ Paradox still offers superior import/export facilities using Lotus 1-2-3, dBASE, ASCII and other file types. It transfers between formats with stunning speed
>
> *Rusel DeMaria, PC Week* ❞

Paradox responds instantly to "Query-by-Example"

The method you use to ask questions is called Query-by-Example. Instead of spending time figuring out *how* to do the query, you simply give Paradox an example of the results you're looking for. Paradox picks up the example and automatically seeks the fastest way of getting the answer. Paradox, unlike other databases, makes it just as easy to query multiple tables simultaneously as it is to query one.

*Source: Software Digest**

Software Digest Rating	Overall Evaluation	Program Name	Version Tested	Ease of Learning	Ease of Use	Error Handling	Performance	Versatility	Memory Requirement	Price
☆☆☆☆	8.7	Paradox	1.1	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	512K	$495
☆☆☆☆	8.2	XDB	1.10	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	320K	$750
☆☆☆	7.6	PowerBase	2.3	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	5.0 - 6.9	UNDER 5.0	384K	$349
☆☆☆	7.0	Open Access II	2.0	5.0 - 6.9	5.0 - 6.9	7.0 - 9.9	7.0 - 9.9	7.0 - 9.9	256K	$395
☆☆☆	7.0	DataEase	2.5/2	7.0 - 9.9	7.0 - 9.9	UNDER 5.0	7.0 - 9.9	7.0 - 9.9	384K	$600
☆☆	6.6	dBASE III PLUS	1.1	5.0 - 6.9	5.0 - 6.9	7.0 - 9.9	7.0 - 9.9	5.0 - 6.9	384K	$695
☆☆	6.4	R:BASE System V	1.1	5.0 - 6.9	5.0 - 6.9	7.0 - 9.9	UNDER 5.0	7.0 - 9.9	512K	$700

RATINGS KEY
(On a scale of 0 to 10)
Overall Evaluation

☆☆☆☆☆ 9.0 or higher
☆☆☆☆ 8.0 - 8.9
☆☆☆ 7.0 - 7.9
☆☆ 6.0 - 6.9
☆ 5.0 - 5.9

All Other Ratings

■ 7.0 - 9.9
■ 5.0 - 6.9
■ UNDER 5.0

Now's the time for a *fast* decision: Upgrade now to 4.0!

BORLAND

Compatibility with Turbo Pascal 3.0

We've created 4.0 to be highly compatible with version 3.0 and included a conversion program and compatibility units to help you convert all your 3.0 programs to 4.0.

Highlights of Borland's new Turbo Pascal 4.0

- Compiles 27,000 lines per minute
- Supports >64K programs
- Uses units for separate compilation
- Integrated development environment
- Interactive error detection/location
- Includes a command line version of the compiler

4.0 also

- Saves output screen in a window
- Supports 25, 43 and 50 lines per screen
- Generates MAP files for debugging
- Has graph units including CGA, EGA, VGA, MCGA, 3270 PC, AT & T 6300 & Hercules support
- Supports extended data types (including word, long integers)
- Does smart linking
- Comes with a free revised MicroCalc spreadsheet source code

4.0 is all yours for only $99.95

Sieve (25 iterations)

	Turbo Pascal 4.0	**Turbo Pascal 3.0**
Size of Executable File	2224 bytes	11682 bytes
Execution speed	9.3 seconds	9.7 seconds

Sieve of Eratosthenes, run on an 8MHz IBM AT

Since the source file above is too small to indicate a difference in compilation speed we compiled our GOMOKU program from Turbo Gameworks to give you a true sense of how much faster 4.0 really is!

Compilation of GO.PAS (1006 lines)

	Turbo Pascal 4.0	**Turbo Pascal 3.0**
Compilation speed	2.2 seconds	3.6 seconds
Lines per minute	27,436	16,750

GO.PAS compiled on an 8 MHz IBM AT

60-Day Money-Back Guarantee**

BORLAND
INTERNATIONAL

For the dealer nearest you or to order call
(800) 543-7543.

CIRCLE NO. 95 ON READER SERVICE CARD

Interlocking Pieces:

Blaise and Turbo Pascal.

Now, for Turbo Pascal 4.0!

Whether you're a Turbo Pascal expert or a novice, you can benefit from using professional tools to enhance your programs. With Turbo POWER TOOLS PLUS™ and Turbo ASYNCH PLUS,™ Blaise Computing offers you all the right pieces to solve your 4.0 development puzzle.

Compiled units (TPU files) are provided so each package is ready to use with Turbo Pascal 4.0. Both POWER TOOLS PLUS and ASYNCH PLUS use units in a clear, consistent and effective way. If you are familiar with units, you will appreciate the organization. If you are just getting started, you will find the approach an illustration of how to construct and use units.

◆**POWER TOOLS PLUS** is a library of over 180 powerful functions and procedures like fast direct video access, general screen handling including multiple monitors, VGA and EGA 50-line and 43-line text mode, and full keyboard support, including the 101/102-key keyboard. Stackable and removable windows with optional borders, titles and cursor memory provide complete windowing capabilities. Horizontal, vertical, grid and Lotus-style menus can be easily incorporated into your programs using the menu management routines. You can create the same kind of moving pull down menus that Turbo Pascal 4.0 uses.

Control DOS memory allocation. Alter the Turbo Pascal heap size when your program executes. Execute any program from within your program and POWER TOOLS PLUS automatically compresses your heap memory if necessary. You can even force the output of the program into a window!

Write general interrupt service routines for either hardware or software interrupts. Blaise Computing's unique intervention code lets you develop memory resident (TSRs) applications that take full advantage of DOS capabilities. With simple procedure calls, "schedule" a Turbo Pascal procedure to execute either when pressing a "hot key" or at a specified time.

◆**ASYNCH PLUS** provides the crucial core of hardware interrupts needed to support asynchronous data communications. This package offers simultaneous buffered input and output to both COM ports, and up to four ports on PS/2 systems. Speeds to 19.2K baud, XON/XOFF protocol, hardware handshaking, XMODEM (with CRC) file transfer and modem control are all supported. ASYNCH PLUS provides text file device drivers so you can use standard "Readln" and "Writeln" calls and still exploit interrupt-driven communication.

The underlying functions of ASYNCH PLUS are carefully crafted in assembler and drive the hardware directly. Link these functions directly to your application or install them as memory resident.

Blaise Computing products include all source code that is efficiently crafted, readable and easy to modify. Accompanying each package is an indexed manual describing each procedure and function in detail with example code fragments. Many complete examples and useful utilities are included on the diskettes. The documentation, examples and source code reflect the attention to detail and commitment to technical support that have distinguished Blaise Computing over the years.

Designed explicitly for Turbo Pascal 4.0, Turbo POWER TOOLS PLUS and Turbo ASYNCH PLUS provide reliable, fast, professional routines— the right combination of pieces to put your Turbo Pascal puzzle together. **Complete price is $129.00 each.**

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

CIRCLE NO. 91 ON READER SERVICE CARD

THE BLAISE MENU

Turbo POWER SCREEN *$129.00*
NEW! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. Now for Turbo Pascal 4.0, soon for C and BASIC.

Turbo C TOOLS *$129.00*
Full spectrum of general service utility functions including: windows; menus; memory resident applications; interrupt service routines; intervention code; and direct video access for fast screen handling. For Turbo C.

C TOOLS PLUS *$129.00*
Windows; menus; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. Specifically designed for Microsoft C 5.0 and QuickC.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM file transfer. For Microsoft C and Turbo C or MS Pascal.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

KeyPilot *$49.95*
"Super-batch" program. Create batch files which can invoke programs and provide input to them; run any program unattended; create demonstration programs; analyze keyboard usage.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

RUNOFF *$49.95*
Text formatter for all programmers. Written in Turbo Pascal; flexible printer control; user-defined variables; index generation; and a general macro facility.

TO ORDER CALL TOLL FREE 800-333-8087

TELEX NUMBER - 338139

YES! Send me the right pieces!
Enclosed is $______ for ______ copies of ______.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: ______ Phone: (____) ____ - ____
Address: ______
City: ______ State: ______ Zip: ______
VISA or MC#: ______ Exp. Date: ______

Microsoft and QuickC are registered trademarks of Microsoft Corporation. Turbo Pascal is a registered trademark of Borland International.

CONTENTS

VOLUME 13, ISSUE 5

ARTICLES

Designing Software ▶

Developing for the User **18**
by Robert Carr
Developing good software is more than just programming and debugging. Carr's eight axioms of successful software delivery tell you the rest of the story.

TIFF ▶

Handling Image Files with TIFF **26**
by Anthony Meadow, Rocky Offner, and Michael Budiansky
If you are writing an application that works with bitmapped images, consider TIFF. . . .

C Programming ▶

Virtual Array in C **46**
by Mark Tichenor
Data arrays need not be limited by available memory. Mark uses virtual arrays with background file management to overcome this problem.

REVIEWS

EXAMINING ROOM **122**
coordinated by Ron Copeland
Products examined from the programmer's perspective. This month's offering include PC/Forms tested in C, Soft-ICE, which provides the capabilities of an in-circuit emulator via software, and DE, a "stretched" version of the standard EMAC editor.

COLUMNS

C CHEST **72**
by Allen Holub
Allen discusses the benefits of compiling to an intermediate language (postfix)—developing for multiple hardware platforms, merging compiler output, and eliminating redundant code.

TO THE MACS **82**
by Stan Krute
Stan takes a field trip to the recent MacWorld Expo. He also has a wrap-up of the Scouting Toolkit.

Pascal ▶

STRUCTURED PROGRAMMING **92**
by Kent Porter
With the help of a special model file—which Kent developed specifically for this exercise—he will demonstrate how to write a Pascal program that reads non-Pascal files.

Parallel Processing ▶

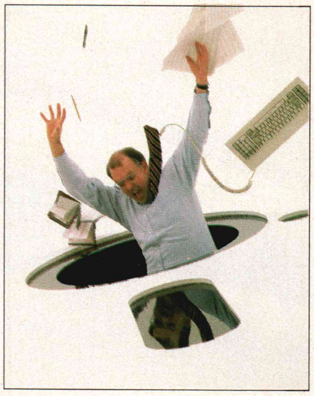
PROGRAMMING PARADIGMS **100**
by Michael Swaine
This is the premier of a new column: Parallel Processing, Object-Oriented Programming, and a reading list. Enjoy!

FORUM

EDITORIAL **6**
by Jonathan Erickson
RUNNING LIGHT **8**
by Tyler Sperry
LETTERS **12**
by you
CARTOON **14**
by Joe Sikoryak
SWAINE'S FLAMES **144**
by Michael Swaine

PROGRAMMER'S SERVICES

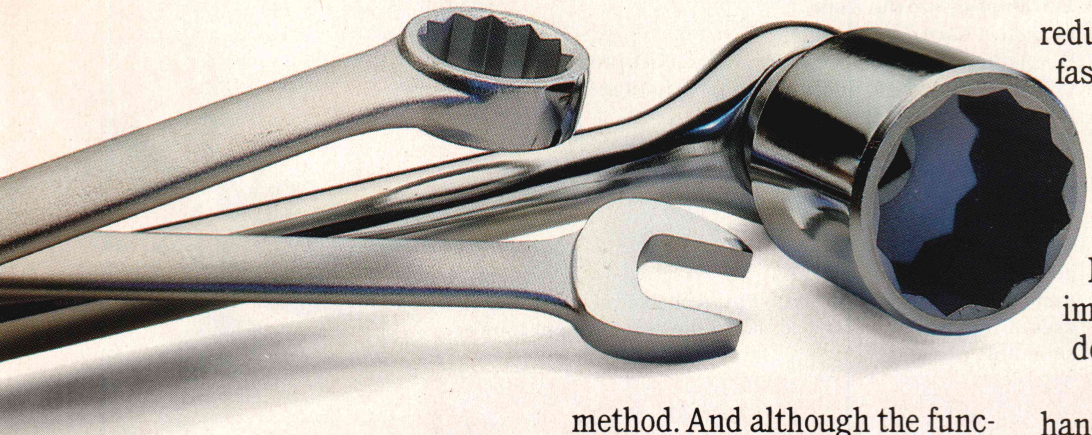
OF INTEREST **136**
brief product descriptions
AD INDEX **137**
where to go for more information on products



About the Cover
A successful conclusion to any journey means that you have to know where you're going, including which pitfalls to avoid along the way if you're going to arrive safely. These same conditions apply to the goal of developing software that is both well designed and well received. Here's hoping this month's modest contributions help you to avoid those nasty pits of alligators scattered along the way.

Next Issue
In June, C-columnist Allen Holub stalks the wild memory allocator, Mike Swaine reports on a recent gathering of programming prognosticators, and, in general, DDJ examines real-time operating systems, dedicated and otherwise.

Buy Our Tools, And We'



Introducing Emerald Bay. The breakthrough database server technology for developing single and multi-user applications. Emerald Bay provides your programs a common data storage and retrieval method which allows data to be transparently shared across multiple and diverse applications.

And when you buy one of our tools for "C," dBASE™ or Lotus® developers, we'll give you the personal engine—free. No royalties to pay, no licenses to sign.

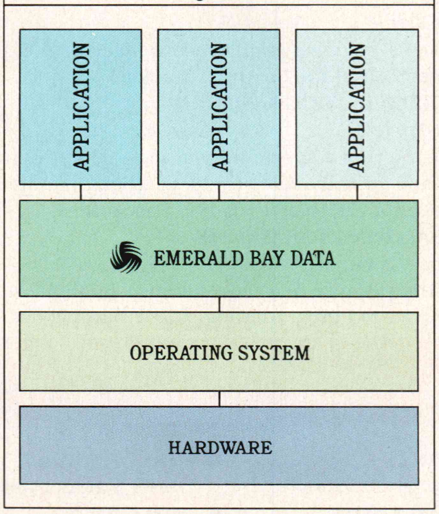
Developed by Wayne Ratliff, the creator of dBASE, Emerald Bay is much more than just another DBMS product, it's an entirely new way to manage data. It's designed to provide an open platform for developing applications in several languages and environments, while Emerald Bay maintains data security, concurrency and integrity.

How The Engine Works

Before, data couldn't be readily shared between applications. But with Emerald Bay, PC applications each share a common data storage and retrieval method. And although the functions of the applications may vary widely, any one application can share another's data transparently; there is no data conversion or translation necessary.

When a PC is an intelligent workstation on a LAN, the Emerald Bay database server technology controls all data security and integrity, including transaction logging with rollback. An application simply makes a request, which is sent to the engine. There, only the essential data is sent back to the workstation. The result is vastly reduced network traffic and faster data access times.

Emerald Bay Architecture



How You Work With The Tools

With the tools we provide, you can easily develop Emerald Bay applications immediately in your familiar development environment.

Emerald Bay technology handles the usually code-intensive management of data, so you can concentrate on what you do best—developing applications.

The ***Developers Toolkit for "C"*** includes well-documented, easy to use "C" libraries that give you the power to create advanced applications, without the effort usually associated with designing and coding a database "backend."

Eagle is Emerald Bay's sophisticated dBASE-like programming language. As the logical evolution of database language, Eagle introduces advanced features, routines and language components, including a compiler, network commands, user-defined functions in "C" and Assembly and automatic index maintenance.

Summit is an "add-in" database management system for Lotus 1-2-3, which gives you sophisticated data manipulation and analysis commands. All three of Emerald Bay's development tools come with the Core Components which include Report Writer, Forms Generator,

I'll Give You The Engine.

an Import/Export facility and the Database Administrator.

The ***Emerald Bay Database Server*** is the heart of the multi-user Emerald Bay technology. Its client/server architecture is superior to current implementations of LAN/DBMS products, and increases total system throughput, while reducing network traffic and access times.

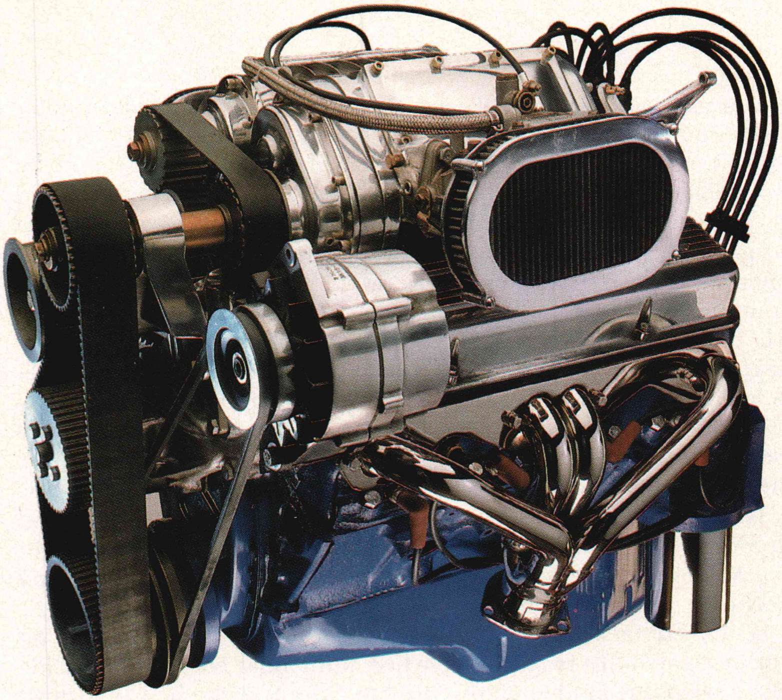
Finally, while providing a path to other operating systems such as OS/2, Macintosh and UNIX, Emerald Bay is a microcomputer-based technology that optimizes your *current* hardware investment.

Free Technical Seminars

We're hosting a series of free Emerald Bay Technical Seminars during April and May in cities across the country. It's your chance to see Wayne Ratliff demonstrate the capabilities of Emerald Bay in person, as well as get some practical experience with the technology yourself.

Call us toll-free at **1-800-777-2027** (and ask for Sandra) for the date and location of the seminar nearest you. Space is limited, so be sure to reserve your seat today.

Emerald Bay. Advanced database server technology. Available *now.*



Emerald Bay Engine Specifications

Data Storage

- Max. databases — No limit
- Max. tables per database — 1000
- Max. fields per table — 800
- Max. field width — 512 characters
- Max. records per table — No limit
- Max. width of records — 10,000 bytes (no limit on ext. fields)
- Max. open databases — 7 (MS-DOS limitation)

Index Storage

- Composite keys supported
- Mixed data type keys allowed
- Keys of up to 100 bytes in length
- Automatic index maintenance
- Ascending and descending keys
- Case independent keys
- Automatic table indexing on record number

Security And Integrity Features

- Access permissions by Read, Write, Delete, Add and Grant
- All five access permissions work on tables and objects
- Read, Write and Grant access permissions operate at field level
- All data other than binary fields can be encrypted
- Transaction logging, with commit and rollback functions
- Full security functions at field and table level
- Optional data encryption at field level

System Requirements

- MS-DOS 3.1 or greater
- Network database server or Single-user computer: PC XT, AT, PS/2 or 386 compatible, 640K, Hard Disk
- Workstation on LAN: PC, XT, AT, PS/2 or 386 compatible, 640K
- NetBIOS compatible networks supported

Trademarks pending: Emerald Bay, Eagle, Summit (Migent, Inc.)

EMERALD BAY

MIGENT™

865 Tahoe Blvd., Call Box 6, Incline Village, NV 89450

CIRCLE NO. 164 ON READER SERVICE CARD

EDITORIAL

Four years ago this month, Mike Swaine wrote his first editorial for *Dr. Dobb's Journal* and in it he talked about *change*—changes at the magazine, changes in the microcomputer industry, changes in general. As it turns out, *change* is the topic of this month's editorial too. You may have in fact already noticed one change. Mike has a new role with the magazine, having moved from the position of editor-in-chief to that of editor-at-large. This gives him a better opportunity to do more original writing, something he's wanted to do for a long time.

For me, Mike's change means a new opportunity. I am Jon Erickson, a former senior editor at *BYTE* and now editor-in-chief of this magazine. *DDJ* has long been one of the microcomputer magazines I've most admired and the chance to lead it into the coming years was a challenge I didn't have to think twice about accepting.

Mike's move doesn't mean that *DDJ* is going to become a radically different magazine from what it has been. I'm as committed to *DDJ*'s original spirit as Mike, and any changes that do take place will be dictated by technological advances and other significant trends that are important to you.

Just coincidentally, the changes at *DDJ*'s parallels some interesting shifts in the microcomputer industry itself. Advances in small system hardware architecture are making life miserable for manufacturers of expensive minicomputers and the emergence (finally) of more sophisticated operating systems like OS/2 promisespowerful new applications. Over the coming months, *DDJ* will be paying particular attention to how those changes affect the software development process.

This changing of the guard, so to speak, provides you with the opportunity to have a say in what directions *DDJ* will go in the future. Take a few minutes to drop me a letter about what you think *DDJ* should be doing and where it should be going. What are we doing right or wrong? What could we be doing better? What aren't we covering that we should be? If you have a specific article in mind that you'd like to write, let me know about it too.

One change I wouldn't mind seeing in the microcomputer industry is a shift away from the tendency of companies to pour resources into litigation instead of R&D. First it was the Lotus look-and-feel suit and more recently Apple's copyright infringement suit against Microsoft and Hewlett-Packard.

There's little any of us can do when influential companies decide to fight it out in the courts instead of on store shelves. But in the meantime, a lot of small (and large) software developers are caught in the middle, wondering whether or not they should sink more resources into further development of Windows applications. In truth, there's probably some merit to the suggestion that part of Apple's strategy is to delay and stifle the development of graphical user interface Windows-based applications. If so, then third-party developers should resent being manipulated like pawns on a corporate chessboard. Maybe by the time this editorial appears, the dispute will be resolved but I don't think so. In the meantime, it looks like it will be the lawyers who profit at the expense of independent developers and end users.

Jonathan Erickson

Jonathan Erickson
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Jonathan Erickson*
Editor *Tyler Sperry*
Managing Editor *Monica E. Berg*
Associate Editor *Ron Copeland*
Assistant Editor *Sara Noah Ruddy*
Technical Editors *Allen Holub*, *Richard Relph*, *Kent Porter*
Contributing Editors *Stan Krute*, *Martin Tracy*
Copy Editor *Rhoda Simmons*
Editor-at-Large *Michael Swaine*

Art/Production
Director *Larry L. Clay*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Barbara Mautz*
Typesetter *Mary E. Lopez*
Cover Photographer *Michael Carr*

Circulation
Circulation Director *Maureen Kaminski*
Fulfillment Coordinator *Francesca Martin*
Subscription Supervisor *Kathleen Shay*
Newsstand Coordinator *Sarah Frisbie*

Administration
Vice President of Finance and Operations *Kate Wheat*
Business Manager *Betty Arsene*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura DiLazzaro*

Marketing/Advertising
Director *Ferris Ferdon*
Advertising Coordinator *Patricia Albert*
Account Managers *see page 137*

Publisher
Peter Hutchinson

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly, with two special issues per year by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; 415-366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Associate Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Request: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 3713, Escondido, CA 92025. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA 800-321-3333; in CA 619-485-9623 or 566-6947. For book/software order problems call 415-366-3600.

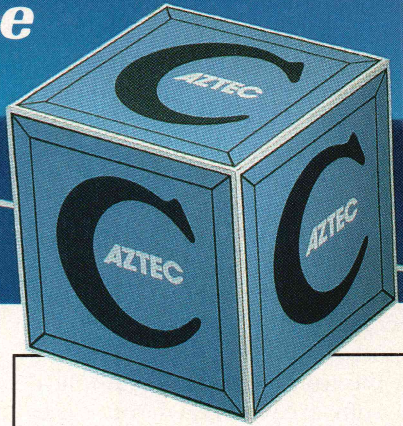
Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $28 per year airmail or $11 per year surface. All other countries add $32 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; 212-686-1520 TELEX 620430 (WUI).

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C. F. von Quadt*
President *Laird Foshay*
Vice President of Publishing *William P. Howard*

Aztec C

Power to go the distance...
Whatever that distance might be



From real time embedded applications to comprehensive commercial applications on Macintosh, IBM PC, Amiga, Atari, and others, Aztec C has earned a well-deserved reputation as an innovative, tough to beat, rock-solid C development system.

But don't just take our word for it—try it yourself. We know that the best way to understand what puts you ahead with Aztec C is to use it. That's why Aztec C systems purchased directly from Manx come with a 30-day, no questions asked, satisfaction guarantee. Call for yours today.

We can also send you information that details the special features and options of Aztec C. Plus information on support software, extended technical support options, and all of the services and specialized support that you may need when you're pushing your software to the limits and . . . beyond.

MS-DOS Hosted ROM Development Systems

Host + Target: $750 Additional Targets: $500

Targets:
- 6502 family
- 8080-8085-Z80-Z180-64180
- 8088-8086-80186-80286/8087-80287
- 68000-68010-68020/68881

Components:
- C compiler for host and target
- Assembler for host and target
- linker and librarian
- Unix utilities make, diff, grep
- Unix vi editor
- debugger
- download support

Features:
- Complete development system
- Fast development times
- Prototype and debug non-specific code under MS-DOS
- Compilers produce modifiable assembler output, support inline assembly, and will link with assembly modules
- Support for INTEL hex, S record, and other formats
- source for UNIX run time library
- processor dependent features
- source for startup

Aztec C Micro Systems

Aztec C is available for most microcomputers in three configurations: The Professional; The Developer; and The Commercial system. All systems are upgradable.

Aztec C68k/Am . . . . Amiga
source debugger–optional

Aztec C68k/Mac . . . Macintosh
MPW and MAC II support

Aztec C86 . . . . MS-DOS
source debugger • CP/M libraries

The following have special pricing and configurations. Call for details.

Aztec C68k/At . . . . . Atari ST
Aztec C80 . . . . CP/M-80
Aztec C65 . . . . Apple II & II GS

Standard System $199
- C compiler
- Macro Assembler
- overlay linker with librarian
- debugger
- UNIX and other libraries
- utilities

Developer System $299
- all Standard System features
- UNIX utilities make, diff, grep
- UNIX vi editor

Commercial System $499
- all Developer features
- source for run time libraries
- one year of updates

MANX

Aztec C is available on a thirty-day money back guarantee. Call now and find out why over 50,000 users give Aztec C one of the highest user-satisfaction ratings in the industry.

Call 1-800-221-0440

In NJ or outside the USA, call 201-542-2121
Telex: 4995812 Fax 201-542-8386

C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724

CIRCLE NO. 150 ON READER SERVICE CARD

RUNNING LIGHT

The original theme for this issue was "Designing Applications," and while I still think it's a perfectly reasonable theme I must confess to some surprise at the variety of responses we got from authors. Space limitations forced us to trim the number of articles more than we would have liked, but that's a chronic problem with magazines and is certainly nothing to distinguish this issue. Suffice to say that we are not finished with the topic of designing applications software with this month's articles.

Something that does distinguish this month's issue is Robert Carr's article on guidelines for developers. After reading the article, you might be tempted to dismiss the message—developing for the user, after all, is something that everyone claims to do. Despite the apparent simplicity and familiarity of Carr's advice, I strongly urge you to read the article and then reconsider your development efforts. A careful examination of your current process might reveal a few holes.

It's tempting to dismiss another reminder of the importance of the user in software design. I might even agree (on a good day) that we all learned that lesson years ago, except for the overwhelming evidence to the contrary.

Item: Just a few months ago, a major software house was in the final beta pass for a product with an integrated editor. It was only at this last pass that someone at a beta site complained that the editor was abysmally slow. The reviewer's comments—something about the seasons moving faster than the cursor—caught the developers by surprise: the program had always seemed fast enough on their 386 machines. Alas, the target audience consisted primarily of people using 8088 machines.

Nor are these problems limited to performance issues. A year and a half ago I pretty much trashed Zoomracks in a review that appeared (substantially rewritten) in *BYTE*. The keyboard interface was a Byzantine complex of Alt- and Control-key commands, the mouse interface—what there was of it—was buggy as hell, and the file facilities could be charitably described as primitive. (The PC version, for example, couldn't recognize subdirectories.)

The capper to this second story is that Paul Heckel, the designer of Zoomracks, is also the author of *The Elements of Friendly Software Design* which is a pretty good book on designing applications.

These aren't isolated instances. There isn't a month that goes by that I don't examine a new program that seems hastily thrown together. Little touches like using subdirectories and environmental variables really aren't that hard to add. And besides, we all know better. End of sermon.

This month's news is probably already old to you: Jon Erickson, our new editor-in-chief, is online starting this month. You can meet him back on page 6.

This month's reading list is a short one. Go to the local university library and pull out George A. Miller's classic 1956 paper, "The Magical Number Seven, Plus or Minus Two." Then consider the implications of human bandwidth on your next design. A little thought might make a world of difference for your customers.

Tyler Sperry

Tyler Sperry
editor

ARCHIVES

Parsimony

"A software development project attempts to solve a problem by producing a solution which includes a software system. Any acceptable solution must meet a set of functional requirements and a set of constraints. Constraints are negative requirements (e.g., compatibility, performance, interface specifications). The more requirements a project has, the more difficult it will be to meet them all.

A particular problem has some fixed requirements and many flexible ones (i.e., unspecified or loosely specified). When a system gets designed for execution on a computer, additional constraints must be imposed (e.g., sequential execution, control structures). Additional constraints are imposed when it is implemented (e.g., finite resources, discrete arithmetic). These constraints are 'artificial' in that they did not come with the original problem.

It is important not to impose any more artificial constraints than necessary in order to aid productivity. Then more room is left for trade-offs which can produce a more desirable result. These views come directly from the belief that simplicity is better than complexity. . . ."—*Kim Harris, "The FORTH Philosophy,"* DDJ, *September 1981.*

Syntactic Sculpting

"A hacker is an artist, and computer artistry is not distinguished from other art forms except in the medium chosen. . . .

"It is the nature of great art that it compresses a great deal of design, intellectual sweat and individual personality into the created object. There are no great paintings painted by a committee or artists. Curiously, it seems to matter little what tools the artist had, or where his starting point was, provided that his accomplishment from that point exceeds what the rest of us could have done. Ansel Adams had color and motion available, but he chose to limit himself to black and white stills—and what art he created! A virtuoso on a violin produces art; the same sound from a Moog is ho-hum. . . .

". . .Unlike painting, sculpture, music, and the like, few people can really appreciate the artistry in a computer product. Perhaps that will change."—*Tom Pittman, "Festschrift for Doctor Dobb,"* DDJ, *February 1985.*

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

IBM PC-BASED

ANALYST WORKBENCH

• Modular • Flexible • Affordable

Your pipeline to productivity in software development, design and documentation

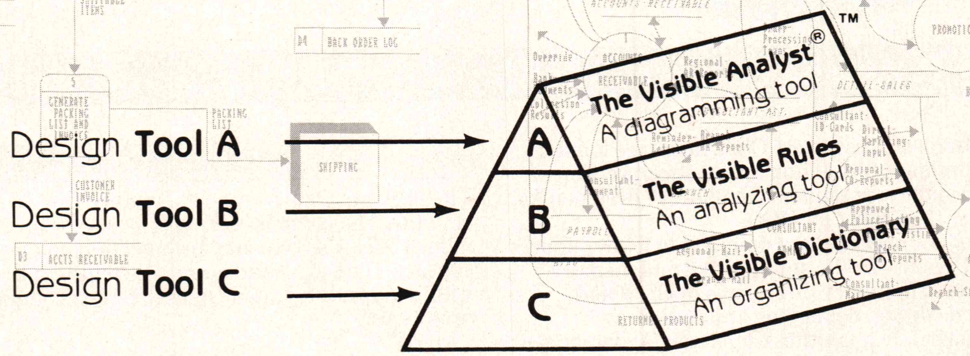
BUILD SYSTEMS FASTER

Priced for your productivity
One for every analyst

$595.00 per tool
quantity discounts

Featuring —

THE VISIBLE ANALYST® WORKBENCH



Software Tools for Software Professionals

FEATURES:

1. Mouse-driven graphics allow analyst to build and test models quickly and accurately. Graphics package includes reuseable custom symbols, reuseable constructs, sizing and stylizing of symbols, variety of line styles and terminators, variety of character sets including bold, italics and sideways. Text sizable for large titles and labels, intelligent graphics object oriented editing, nesting, paging, export/import, 180 dpi high quality output...and more. Designed to handle all of the leading structured methodologies as well as diagramming conventions that require special symbols or configurations.
2. Expert Rules System allows for menu selection of leading structured methodologies. Yourdon, Gane and Sarson supported. Specialized or Custom Rules can also be implemented. Validates and balances data flow diagrams over any number of levels...and more.
3. Key information automatically loaded into data dictionary. This information can describe processes, data elements, data records, various files, charts, narratives, objects, etc. Provides a cross index, global or wild card search, replace and delete items, checks for aliases, does audit trail and produces reports in various formats...and more.
4. Low cost provides you with standards throughout your department or company - better communications means better systems faster...start mapping your information today.

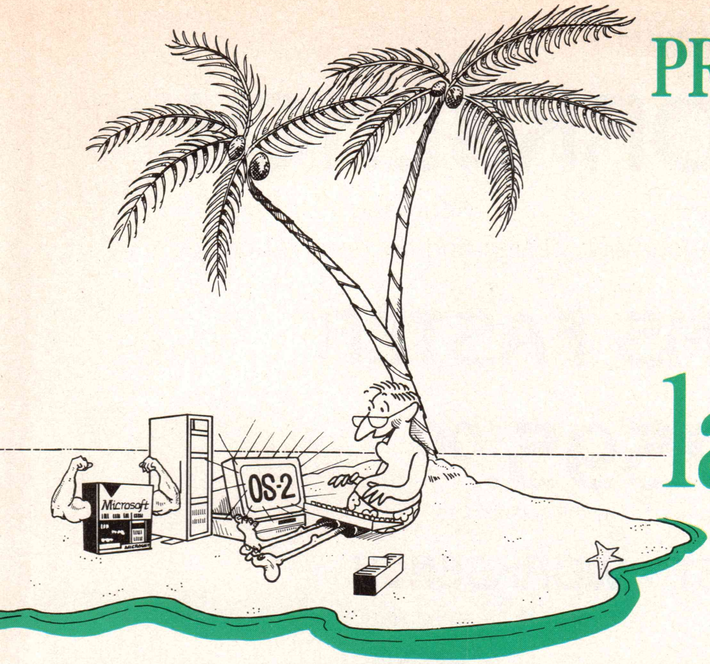
For more information contact your nearest dealer or distributor or call, write, or telex.

CALL 617-969-4100 or Telex 261102

Start Experiencing THE VISIBLE SOLUTION™ Today!!!!!

VISIBLE SYSTEMS CORPORATION

CIRCLE NO. 251 ON READER SERVICE CARD



PROGRAMMER'S PARADISE PRESENTS

Strong language from Microsoft®

Microsoft & Programmer's Paradise offer you the best value for the Tools you need

They're here…Microsoft's complete OS/2 language family, a new "smart" programmer's text editor, an enhanced version of CodeView debugger and Programmer's Toolkit. All designed for OS/2 development with support for DOS.

The language product upgrades are designed to let developers write programs that access OS/2's multitasking and memory-management features. They allow direct calls to OS/2 and include an integrated editing, debugging and compiling environment that can run in a multitasking mode. All languages can break the 640K barrier, create both protected mode and real mode programs, and support "bound" programs. They also include the upgraded CodeView debugger which now handles .exe files up to 128 megabytes.

Microsoft's new program editor is included in all five languages and provides a complete development environment. From within the editor, the user can compile, link, debug compile errors, and even start a series of compilations using a MAKE file. This editor is the first full-featured programmer's editor available for the protected mode of the OS/2 operating environment.

New Programmer's Toolkit

Contains three reference manuals for a complete description of system functions, structures and file formats. Special utilities to help develop OS/2 applications. In addition to the documentation and software, the toolkit includes two free hours of support via Microsoft's electronic-mail product support system.
List: $350 **Ours $229**

Take advantage of NEW BASIC Compiler/6.0

Gives you more control over portions of the run-time library which are linked to your program enabling creation of smaller stand alone executables. QuickBASIC is included for an instant development environment. Exceptional debugging power of enhanced CodeView. Alternate fast math library for programs without a math coprocessor. Advanced language features such as user defined types, recursion and huge arrays.
List: $295 **Ours $189**

Experience the power of *NEW* Pascal Compiler/4.0

Now bolstered by CodeView for quick and efficient debugging. Compile any standard ISO or ANSI program. Meet target requirements with your choice of math options. Link and edit with greater efficiency with Microsoft's new incremental linker.
List: $300 **Ours $189**

C Optimizing Compiler/5.1

The ultimate C environment. Produces the fastest code available on a PC. The compiler converts nearly 20 function calls into inline code. Also includes optimizations such as loop invariant expression removal and automatic register allocation of variables within loops. Integrated QuickC for fast compilation and prototyping.
List: $450 **Ours $285**

Macro Assembler/5.1

Is the bridge you've been waiting for! Puts all the speed and power of assembly-language programming within easy reach. Simplified segment directives allow easy program and subroutine setup. Has an assembly rate of 25,000 lines per minute. Special constructs make the writing of a mixed language routine as simple as identifying the calling language and the parameter to be passed.
List: $150 **Ours $99**

FORTRAN Optimizing Compiler/4.1

Gives you mainframe power and performance on your PC… Highly optimized, compact code for fast running programs. Full and complete implementation of the ANSI 77 FORTRAN standard. Full set of development tools is provided including an: incremental linker, protected-mode text editor, library manager. GSA-certified, error-free.
List: $450 **Ours $285**

1-800-445-7899
In NY: 914-332-4548

Programmer's Paradise™
A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591

CIRCLE NO. 194 ON READER SERVICE CARD

LETTERS

MASM Complaints

Dear *DDJ*,

It's amazing to see the phrase "screamingly fast" applied to the lumbering leviathan that is MASM 5.0 (Examining Room, February 1988). Compare the reviewer's measurement of 120 instructions per second to the greater than 1,000 instructions per second of Eric Isaacson's A86, which assembles directly to executable code and simultaneously creates a symbol file and embeds error messages in the source code. And it does all this quicker than MASM links. I used it to assemble a 12-module program to a .COM file in 4.8 seconds; it took MASM 54 seconds to assemble the same program and 5.7 seconds to link it.

The real purpose of this letter is to complain that MASM 5.0 no longer supports .COM files and nobody says anything about it. CodeView does not offer symbolic debugging of .COM files and MASM 5.0's symbol file is not compatible with Symdeb or other symbolic disassemblers. So even if you're satisfied with the creepy-crawly rate at which MASM 5.0 assembles and links, you have to give up .COM files or symbolic debugging.

I wish you would have a person review assembly-language products who does not think that assembly language is the "most tedious of programming languages" and that "the emphasis is shifting away from assembler as a primary language" and does not call assembly language "assembler." A person who can't be bothered to distinguish verbally between the language and the translator and who thinks that assembly language is unpleasant will not pay attention to details important to assembly-language programmers.

I ask you: In what other language can you write an assembler that assembles 1,000 instructions a second while it does other things and ends up occupying 20K? Or in what other language can you write a utility that searches for a file name in a full 10-Mbyte, 70-subdirectory hard disk in 12 seconds? Why do advocates of one tool (a higher-level language in this case) disparage others?

George Frank
Encinitas, Calif.

Kent Porter responds:
In comparison with earlier releases, MASM 5.0 is "screamingly" faster. There are even faster assemblers, and I'll take George's word that A86 is one of them. It's true that MASM 5.0 doesn't produce .COM files directly; you have to run the end product through EXE2BIN if you want a .COM. It's also true that I didn't mention that, and I apologize on behalf of all us reviewers who haven't said anything about it.

Assembly language is indeed tedious. That doesn't mean it's bad, nor did I suggest that it is. It's a tool and it has its place. Most developers have switched to C, Pascal, or Modula-2 because they're more productive (that is, less tedious) languages, writing only high-overhead routines in assembly language. Thanks, George, for setting us straight on the proper use of terminology. When I started programming IBM mainframes in the early 60s, we called the language "assembler," and I think I overheard some other professional programmers still misusing that same term just last week.

It's a revelation to learn that writing about structured programming makes me an "advocate" in some grandiose struggle for language supremacy. Let's try to keep things in perspective, shall we?

Don't Believe Everything You Read

Dear *DDJ*,

I am embarrassed to say that I must retract two statements I made in my letter published in the November 1987 issue of *DDJ* on the subject of teletypewriter terminals. I was careful to get my facts concerning teletypewriters correct, but I was not as careful with the examples I gave of other standards that had outlived the reasons for the standards being set the way they were. I received a letter from Mr. Clive J. Grant, a professional engineer, debunking both of these examples.

I have forgotten where I read the story of the Roman emperor and the railroad gauge, but it was interesting and plausible and I fell for it. According to this tale, the Roman military had a problem with ruts. Ruts in unpaved roads tend to enforce at least a local standard on wheel spacings because a vehicle with a wheel spacing not matching the rut spacing is in difficulty. The trouble was that in different parts of the Roman Empire, there were different local standards, and this created problems with chariots and other military vehicles when the legions were moved from one place to another in the empire. Therefore, the emperor issued a decree standardizing wheel spacings in the empire, and this standard, enforced by the ruts, endured long after the empire had fallen.

When the railroads were first started, so the story went, the developers turned to the carriage makers for the rolling stock, and this resulted in the standard being transferred to rail spacing. In fact, when Stephenson established the railroad gauge, rail systems were in use in the Cornwall mines and he took an average of the mine rail spacings, which varied widely. There is some evidence that there was a Roman standard for wheel spacing, but it may have applied only to Rome and the immediate vicinity and in any event had no connection with the railroads.

The typewriter keyboard tale came from an article advocating the new keyboard layout that is supposed to be much faster. Mr. Grant points out, however, that Sholes, who originated the QWERTY key-

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

Call 1-800-45-VEDIT for FREE Fully Functional Demo Disk

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS easily satisfies the most demanding computer professional.

Try a Dazzling Demo Yourself.

The free demo disk is fully functional—you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial—you experiment in one window while another gives instructions.

The powerful "macro" programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a "macro"—it shows that no other editor's "macro" language even comes close. And VEDIT PLUS is only 40K in size.

Go ahead. Call for your free demo today, You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Only VEDIT PLUS is this Flexible.

The installation lets you pick from closely emulating the keyboard layout of Word Perfect, WordStar and others. Or you can easily create your own layout and even your own editing functions. Supports any screen size—you pick screen colors and attributes.

Supports the IBM PC, XT, AT and PS/2. Also supports MultiLink, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, FlexOS (protected mode), CP/M-86 and CP/M. (Yes, we support windows on most CRT terminals, including CRTs connected to an IBM PC.) Order direct or from your dealer. $185.

Special: VEDIT (single file, no windows) for CP/M—$49.

- **Fully Network Compatible**
- **Call for XENIX-286 version**
- **30 Day Money-back guarantee**

Compare Features and Speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	Unlimited
Multiple file editing	20 +	2	No	20 +
Windows	20 +	2	No	20 +
Macro execution window	No	No	No	Yes
Pop-up menus	No	No	No	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Automatic processing of Compiler errors	Yes	No	No	Yes
"Cut and paste" buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
Convert to/from WordStar	No	No	No	Yes
On-line calculator	No	No	No	Yes
Configurable Keyboard	Hard	No	Hard	Easy
43 line EGA support	Yes	No	No	Yes
Manual size/index	250/No	42/no	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec

VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. PMATE is a trademark of Phoenix Technologies Ltd. Norton Editor is a trademark of Peter Norton Computing Inc. MultiLink and PC-MOS/386 are trademarks of the The Software Link, Inc. CP/M and FlexOS are trademarks of Digital Research. MS-DOS is a trademark of Microsoft.

*Also available for TI Professional, Tandy 2000, DEC Rainbow, Wyse WY700 and others.
*Demo disk is fully functional, but does not readily write large files.

CIRCLE NO. 109 ON READER SERVICE CARD

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, TELEX 701821

(continued from page 12)

board, never revealed why he arranged the keys in that manner, so any reason advanced for it is speculation. Further examination of this particular speculation shows that its motivation was not to slow the typist, but rather to reduce key pileups.

On the old mechanical typewriters, if a new key was struck before the previous keystroke had retracted, then a pileup occurred. Therefore maximum typing speed was limited by the time required for the spring to retract the previous keystroke after the key had been released. However, many pileups occurred when a new key was struck with a different finger before the previous key had been released. The speculation was that the keyboard was arranged to assign groups of keys apt to be struck in succession to the same finger and thus ensure that one key would be released before the next was struck. This is not quite the same as trying to slow typing.

I hope that my blind acceptance of interesting stories I have read has not caused too much of a problem by misleading your readers.

David S. Tilton
Manchester, N. H.

HyperCard Ends an Era?

Dear *DDJ*,

I appreciated Mike Swaine's comments concerning HyperCard in "Running Light" in the January issue of *DDJ*. Many of his points were right on the mark. Others, however, were wild shots that may sound logical but not to an ol' Mac end-user. HyperCard will indeed bring about a proliferation of stackware, and no doubt there is going to be a lot of sloppy programming. This, I agree, is inevitable. Will it threaten the Mac user interface as you suggest? This I seriously doubt.

HyperCard is merely the growing momentum in computerdom to making the world of computers more "user friendly." Stackware may indeed become polluted for awhile, as everyone with a new Mac jumps into the programming ring. But when all the smoke finally clears, what you will have is the availability of narrowly focused programs that serve a limited consumer base that would not otherwise be met by commercial programmers. The reason? Lack of interest, lack of monetary reward, and most of all lack of knowledge in or about these highly specialized areas.

The authors of many of these stacks will be professional people like myself who have an interest in a particular area and are acutely aware of special needs. Our reward for meeting those needs will transcend monetary gains. We are by nature not "sloppy," particularly in our work or whatever interests us. It doesn't mean we will challenge Microsoft or even rival Danny Goodman's work, but it will be good. Above all it will meet the needs of small groups of people who would otherwise be ignored.

I don't believe HyperCard spells the end of anything. Just as the Macintosh has made computers easier "for the rest of us," so will programs such as HyperCard make producing programs easier. That is as it should be. The progress that has been made in making computers easier to use in both the hardware and software will not only continue but will also accelerate.

Programming will not be immune from this progress. We will see "home videos" in software, but Microsoft or even your local computer store won't be selling them—any more than NBC or your local T.V. channel shows home videos. There will be a lot of amateurish stuff around, but it won't hurt anything. Indeed, it will help. It will stimulate imagination and generate interest. There will be a lot of really good stuff that you will never hear of because of limited interest and distribution. No, HyperCard isn't the beginning of the end, or even the real beginning—that occurred with BASIC or perhaps even before. It is just one more milestone in the evolution of the information era.

Ronald L. Cox
Poplar Bluff, Mont.

" 'Faster running' was last year's marketing plan. We wanted to sell something with a BANG!"

(continued on page 132)

PVCS

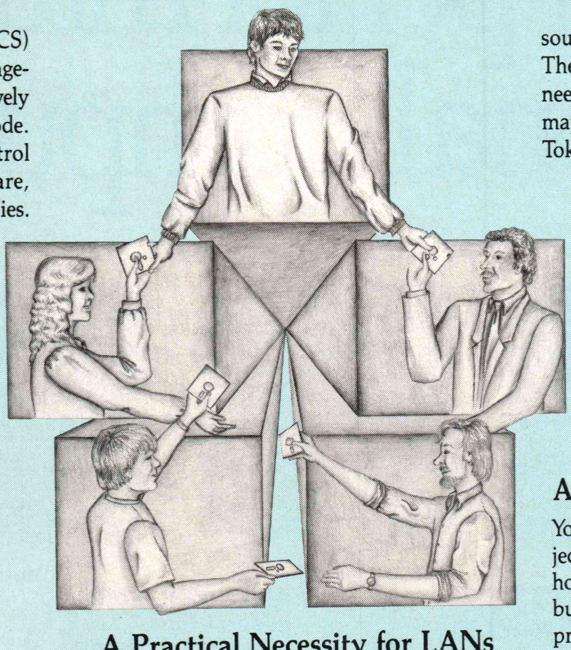
MS-DOS COMMON INTERFACE & FILE FORMAT VAX/VMS

The Number One Source Code Control System.

The POLYTRON Version Control System (PVCS) simplifies and automates Configuration Management so programmers and managers can effectively control the revisions and versions of source code. PVCS is the most widely used change control product and is used by the leading software, aerospace, manufacturing and service companies.

"In terms of features, PVCS provides everything necessary to a large multi-programmer project — more than any other package reviewed. No restrictions are placed in the development environment and all aspects of operation can be customized for specific project needs."

PC Tech Journal
September 1987



Unmatched Flexibility

- Storage & Retrieval of Multiple Revisions of Source Code
- Maintenance of a Complete History of Changes
- Control of Separate Lines of Development (Branching)
- Resolution of Access Conflicts
- Optional Merging of Simultaneous Changes
- Release and Configuration Control
- Project Activity Reports
- Management Reports
- Command or Menu Interface

Project Control

PVCS maintains individual archieves of all project components in your system — source code modules, data files, documentation and even object code libraries. These "source documents" can be written in any language or multiple languages.

Fast Retrieval of Revisions

PVCS uses "reverse delta storage" which saves disk space and speeds retrieval of versions of any file in the project database. A delta is the set of differences between any revision and the previous revision. PVCS can rapidly recreate complete versions of any file whether it is the most recent revision of a module or the original version of the entire project. Differences are automatically detected and stored.

A Practical Necessity for LANs

While important for single-programmer projects, PVCS is absolutely essential for multiple-programmer projects and LAN-based development efforts. In a LAN environment, source code files are simply too easy to change. Because any change to any file can have major ramifications, coordinating and keeping a record of changes is critical. Project leaders can determine, on a module-by-module basis, which programmers can access or modify source files, libraries, object code and other files. The levels of security can be tailored to meet the needs of nearly every project. PVCS works on all major LANs including 3Com, Novell and the IBM Token Ring Network.

"PVCS has helped us maintain nearly 90 programs and utilities. Without it we would not have the quality of our upcoming release of NetWare."

Jonathan Richey
Manager, NetWare Utilities
Novell

Adopt PVCS on Your Existing Projects

You can obtain the benefits for your current project without disrupting development, regardless of how long your project has been under way. You can build PVCS archives from revisions stored in your present files or simply adopt PVCS from the current date.

PolyMake Reads PVCS Logfile Format

PolyMake, the leading Make utility, understands the structure of PVCS logfiles and is able to correctly determine the date and time of any revision. This prevents unnecessary operations that occur when the date and time of the complete project archive itself is used as with other make utilities.

Once you standardize on PVCS, the archives used to track and monitor changes are interchangeable between any PVCS product.

Personal PVCS — Offers most of the power and flexibility of Corporate PVCS, but excludes the features necessary for multiple-programmer projects.

Corporate PVCS — Offers additional features to maintain source code of very large and complex projects that may involve multiple programmers. Includes multi-level branching to effectively maintain code when programs evolve on multiple paths.

Network PVCS — Extends Corporate PVCS for use on Networks. File locking and security levels can be tailored for each project.

PVCS for VAX systems — Requires VMS. Uses the same interface and archive format as MS-DOS version. Supports branching and offers file locking and other security features for multiple-programmer projects.

	MS-DOS*	VMS		
	PC/XT/AT	Micro VAX II	VAX 7xx	VAX 8xxx
Personal PVCS	$149			
Corporate PVCS	$395			
Network PVCS	$995**	$4,950	$9,500	$10,500+
PolyMake	$149			
Network PolyMake	$447**	$1,250	$2,375	$2,500+

**5 Station LAN License. Call for pricing on larger Networks.

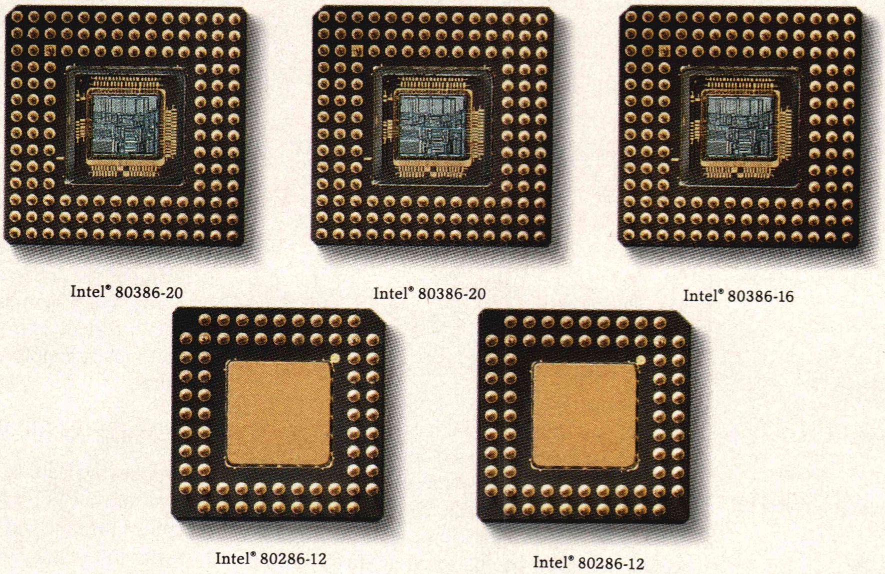
TO ORDER:
1-800-547-4000
Dept. DDJ

Oregon & Outside USA call (503) 645-1150.
Send Checks, P.O.s to: POLYTRON Corporation, 1700 NW 167th Place, Beaverton, OR 97006

®
POLYTRON
High Quality Software Since 1982

CIRCLE NO. 190 ON READER SERVICE CAR

These give you high-performance personal computers.



COMPAQ personal computers offer far more than advanced, high-speed microprocessors. Each offers a combination of innovative features which work with the processor to maximize overall system performance. So there's nothing to slow you down.

Take system architecture, for example. The COMPAQ DESKPRO 386/20 and COMPAQ PORTABLE 386 are built around an advanced 32-bit concurrent bus architecture which exploits the speed of the computers' 20-MHz 80386 microprocessors. Two buses—one for memory and one for peripherals—eliminate information bottlenecks, allowing each component to run at its maximum speed. This ensures the highest system performance without sacrificing compatibility with industry-standard hardware and the world's largest library of business productivity software.

Similar performance enhancements are engineered into each subsystem of every COMPAQ personal computer. Each component is then optimized individually, yet designed to work as part of the total system.

For instance, COMPAQ Fixed Disk Drives deliver both high capacity and high performance. You can install up to a 300-megabyte fixed disk drive in the COMPAQ DESKPRO 386/20 and up to a 100-megabyte drive in the COMPAQ PORTABLE 386. More importantly, you can get to that data almost instantly thanks to some of the industry's fastest access times—averaging less than 30 milliseconds. When you combine this speed and capacity with disk caching, the result is the highest-performance storage subsystem in the industry. To take it one step further, Compaq helps you protect that data with internal high-

These give you the highest-performance personal computers.

speed fixed disk drive tape backup systems.

Another graphic example of Compaq total system performance comes from the COMPAQ Video Graphics System. This system supplies VGA graphics with high-resolution COMPAQ Color and Monochrome Monitors along with speed enhancements from the COMPAQ Video Graphics Controller Board. When the board is used in a 16-bit slot, it makes screen updating 50% faster than the IBM® PS/2™ Video Graphics Array and other comparably equipped systems.

Uncommon performance innovations like these are common to all COMPAQ desktop and portable personal computers. That's clearly why each one is the best in its class, and why together, they represent the most powerful line of personal computers in the world. That's also why Compaq consistently earns the highest performance and quality ratings from computer experts. And unsurpassed satisfaction ratings from computer users.*

Any computer can use a fast microprocessor. But it takes high-performance subsystems surrounding the processor to achieve the highest system performance in the world. The kind achieved by Compaq.

For more information and the location of the Authorized COMPAQ Computer Dealer nearest you, call 1-800-231-0900, Operator 49. In Canada, 1-800-263-5868, Operator 49.

COMPAQ

It simply works better.

Developing for the User

by Robert Carr

How do bad software products come about? The reasons vary, but one of the most common faults is when programmers design the product "for themselves," forgetting the product's real users. The key to avoiding this common pitfall is to realize that programming is ultimately for the user—the consumer of the software. You must view the process of developing software as a user-driven, software development cycle.

Software development is much more than programming and debugging; it's an overall life cycle of delivering software products to users. So central is the relationship between the user and the software product that the overall process should be called software delivery, not software development, emphasizing the delivery aspect as the clear goal.

The cycle of software delivery can be broken up into several phases, each with its own axioms. This article lays out eight such phases and their dictums. From the viewpoint of user-driven development, each phase offers a special contribution to the goal of successfully delivering software that makes your users rave about, demand more of, and pay for your software (and ultimately your salary). Although the eight phases are listed here in their natural order, in reality they often overlap; ideally, there is quite a bit of iteration and looping.

A last prefatory comment—although I refer to project teams, all these comments apply equally to the solo implementation effort.

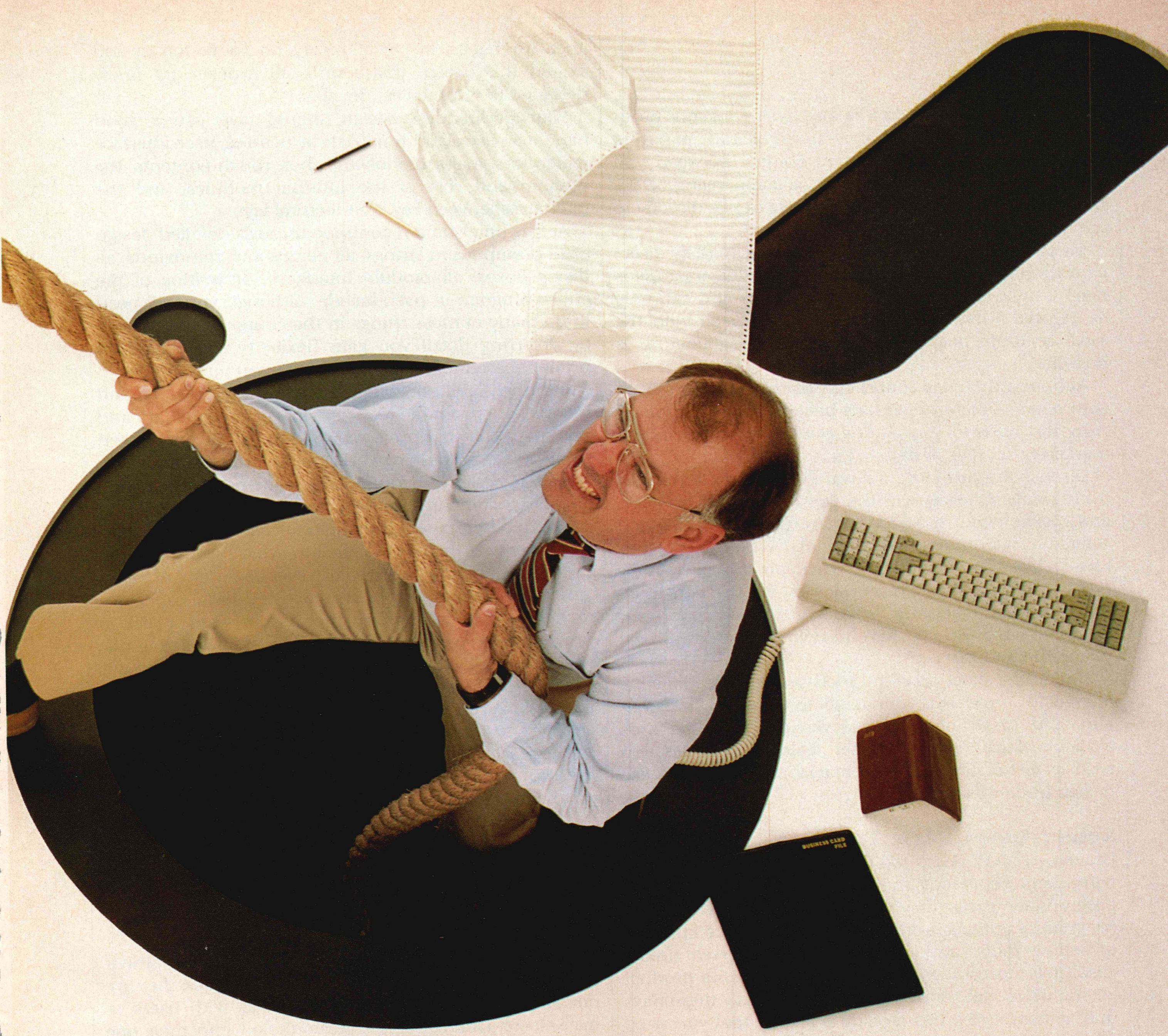
Know Your Users

Just as journalists and writers for decades have offered the sage advice to "know your audience," so should developers know the users of their products.

How can you find your users, so as to know them? Some developers are hired directly (say, as contract programmers or through corporate MIS departments) by their users or users' organizations that will use their products; these developers are fortunate—they know exactly where to find their users. Other developers in some sense develop for "markets" of unknown users; they must find a representative sample of potential users.

Once you've found your users, do something simple with them: talk to them yourself. Talk to

Robert Carr is vice president of software for GO Corp., 139 Townsend St., San Francisco, CA 94107. He is the creator of Framework II and former chief scientist at Ashton-Tate.



Avoiding the common pitfalls of software development is easier with the user's help.

(continued from page 18)

them extensively, and do it before designing your product. If you have the sometimes mixed blessing of having a formal marketing department coming between you and your market/users, insist on accompanying or bypassing the product-marketing people to talk directly to users.

Much useful information can only be gained by direct contact with users. Software that makes users happy is hard to deliver on time and within costs and can only be created when design decisions are made with the most accurate projection of the users' reactions as is possible.

Although this may sound surprisingly simple, talking with users yields tremendous benefits when compared with the costs of presenting your users with formal marketing surveys or detailed product mock-ups.

Discuss the proposed software with them. Have they used similar packages in the past? What did they like and dislike about them? What are their hardware and software environment requirements? When would they ideally like the software by; when do they absolutely need it by? Or, if you're developing for the consumer market, what marketing windows do you have?—for example, when do you have to deliver in order to beat competitors to market?

One of the most powerful tools you have for meeting a targeted ship date is your ability to adjust the functionality and scope of your first version to fit the time available. Early, extensive user dialogue provides you with a gestalt feel for your market so you can make design trade-offs intelligently.

Select Appropriate Tools

Craftsmen need sharp tools. Happily, because personal computers represent the first mass market ever for programmer tools, they're beginning to host the best programming tools ever. Unless you are in the business of selling tools, avoid building them yourself unless absolutely necessary—your job is building the product for the users, not tools for yourself. It's likely these days that someone else has already built the tool you need and is willing to sell you a copy for a few hundred dollars. Buy it and profit.

As with tools, so with building blocks. Seek to partition your architecture such that as many pieces as possible can be subject to the "buy vs. build" decision. Strive to prejudice yourself toward buying. And even when the final version shouldn't be shipped with a preexisting building block, consider using it in the early phases of your project so as to get the overall system running as early as possible. This is a vital technique, supporting the evolutionary-development technique discussed later.

Spend Time on Preliminary Design

Now that you understand the needs of your users, don't rush into programming; instead, spend adequate time designing an architecture that meets all major product requirements, including user interface and performance. Go this far, but by all means do not go too far and extend this design phase until all aspects are finely designed to the smallest details!

Your preliminary design should give you a good handle on key algorithms, data structures, user-interface metaphors, major menus and their rough contents, the number and role of the internal modules, and the interfaces between key architectural layers.

At this point do not continue through detailed design of all prompts and strings, all menus and commands, all dialog boxes, all module interfaces, or writing of the entire program in pseudocode. Although you will need to do many of these things in the course of the project, by deferring detail you gain flexibility to change the product based on what you learn as you build it.

Although much of software engineering research and theory stems from the requirements of large military and aerospace software projects, we in the PC business need to be wary of applying these techniques to our work. What's necessary to produce satellite-control software at a large company is not necessarily the appropriate software development paradigm for us. And nowhere is this more true than in the need to avoid committing the complete design to paper before ever beginning to build.

Now is the time to perform selected tests, especially to highlight major performance issues—run a test to prove your design is sufficiently efficient. If, for example, you wonder if your windowing environment can paint characters fast enough, then write a test program now. If it doesn't, you know you have to redesign your display algorithms to minimize character painting.

The reason behind spending adequate time on design is to put your best efforts into avoiding major surprises or gotchas in the overall design of the product; these are the most costly to rectify. Even with your best efforts, it's likely there will be a couple of medium-size changes of direction/design that only become apparent in midcourse.

Use Your Team's Latent Talent

The most underutilized resources on most development projects are the programmers themselves! Often they are not allowed to discuss their own ideas with users or marketeers. Often they are pigeonholed into their one area and aren't even allowed to contribute to areas outside their own.

Ask discipline, professional work habits, and friendly interpersonal skills of your development team, and couch design and planning sessions in a consensus atmosphere that invites the best from everyone. Your team will then take the project closer to perfection than you guessed was possible. Emphasize the word *team*: what the team should value and aim for is what they can do together, not what any of them achieves individually.

A team-oriented consensus style is in strong contrast to concepts such as chief-programmer-based teams, for example, in which team members exist merely to make the guru chief programmer more efficient, implementing the designs that spring from his head. A benefit of inviting more contributions from team members is that their professional skills will develop more rapidly.

We engineer ours onl with a passion for per

While some companies sell a lot of computers because they make something for everyone, we sell a lot because we don't.

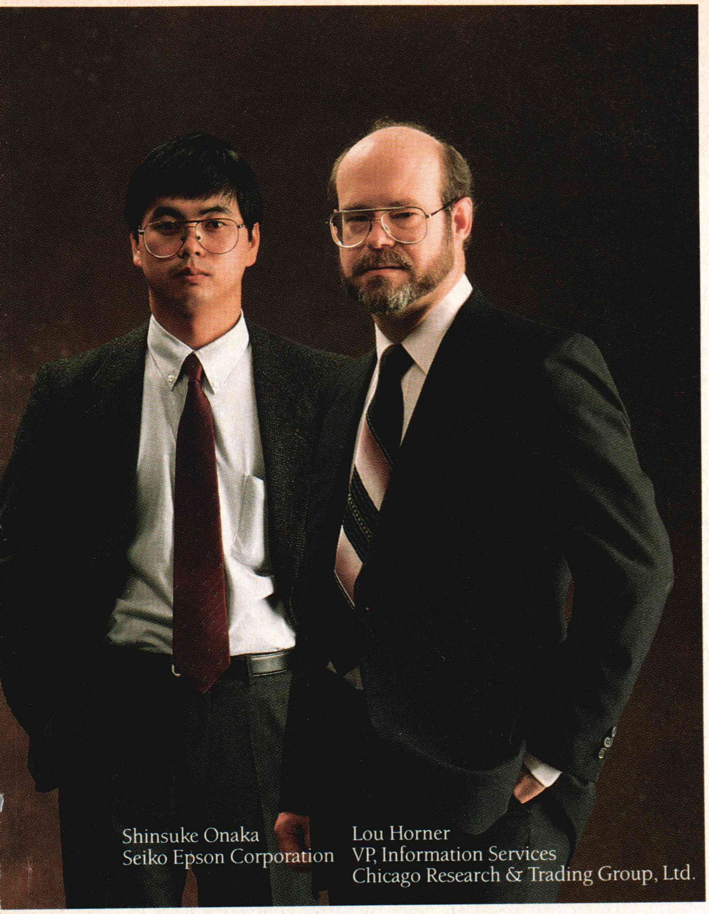
All the workstations we make, the applications that run on them, and the networking power that unites them with the other computers in your company were created for a select group of people.

Namely the engineers, product designers, software developers and other professionals who demand nothing less than ultimate compute performance.

People who clamor for access to processing power and graphics. Who possess an insatiable appetite for information. And who can ill afford to endure the delays, limitations and obstacles that typically hinder the effort to attain it.

If you're such a person, you should have an Apollo workstation. For you'll realize the moment its screen is in front of you that the issue of performance is behind you.

An Apollo workstation will grant upon you enough dedicated compute power to keep your imagination charged permanently. Letting you choose from a compatible family of workstation systems whose prices start as low as a personal computer and whose perfor-



Shinsuke Onaka
Seiko Epson Corporation

Lou Horner
VP, Information Services
Chicago Research & Trading Group, Ltd.

y for those formance.

mance extends to that of supercomputers.

These machines will grant you imagery so brilliant you won't want to blink for fear of missing something. With real time two- and three-dimensional graphics that render up to 16.7 million colors at 130,000 vectors per second. And they'll open your eyes even wider with networking power and elegance.

Every Apollo workstation, from the Series 3000™ Personal Workstation™ to our new Personal Supercomputer,™ functions as a command center from which you have unequalled access to data, processing power, development tools, and applications.

So that every mainframe, minisuper, and microcomputer on your network is at your beck and call.

In a manner almost invisible to you, our workstations show you networking performance you probably thought impossible.

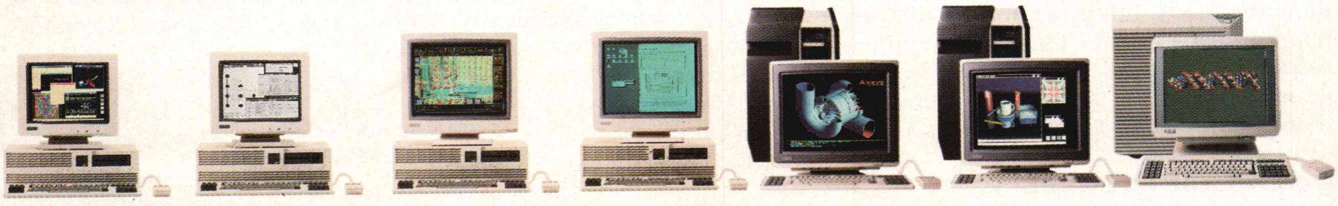
For with the industry's first implementation of Network Computing Architecture,™ they make your multi-vendor network appear as one computing environment.

Letting you run a single application on a network of computers by automatically dispatching portions of a program to the processors most qualified to execute them. And providing the tools to develop and debug code running on different machines.

All while freeing you to create applications, access network resources and even move from one operating environment to another with whatever language, menus and file names you define.

A fact that might inspire you to wonder if we don't engineer our workstations only for you.

Today, there is more than one way to measure computer performance. But when the criteria include processing power, graphics and network computing, nothing measures up to Apollo.



apollo

For more information, call 1-800-323-1846. In Massachusetts call 1-800-847-1011. Or write Apollo, 330 Billerica Road, Chelmsford, MA 01824.

DEVELOPING FOR THE USER
(continued from page 20)

Design and Develop Through Evolution

Once you have a clear understanding of your users' needs and your product design, you're ready to begin programming. It's key now to adapt an evolutionary implementation method that provides you with the flexibility to make changes as you learn from the growing product.

Some day there may be CASE tools that allow an entire product to be changed with ease, but for now the best way to obtain flexibility is to build from the outside in: begin building from the user interface inward and from your low-level building blocks upward. It's the middle levels of most software products that consume your implementation effort and are the hardest to change. For it is at these levels that functionality and behavior tend to be hard-wired into the architecture.

Low-level building blocks tend, on the other hand, to be understandable and flexible no matter what the flow and functionality of the higher levels. At the user-interface level, use scaffolding and other temporary crutches to get your user interface running. By doing so users can interact with the actual product itself, providing valuable polish and validation to your design.

If necessary you can prototype the user interface using prototyping tools such as Dan Bricklin's Demo program. Strive, however, to avoid using such tools for extensive mock-ups when you could spend your time developing the real code (with a little throwaway scaffolding).

Empathize with Users

If ease of learning is an important goal of your project, now is the time to sit a few users down and have them try to learn the system. Watch them yourself and have your programmers do the same (videotaping is a good method here). If three out of four subjects stumble at the same point, it's likely that area needs redesign. Such user testing can be both cheap and easy to perform. (If your market includes computer novices, for example, you can use relatives and friends as test subjects.)

A long-term benefit of having programmers and program designers talk with users and view user testing is that over time the programmers sharpen their "user empathy" skills. Over the course of a couple of projects, the programmers will develop user empathy—the ability to anticipate, during the design phase, many user problems your team will begin to get a reputation for being sharp user-interface designers.

Many programmers resist changes to their initial designs suggested by the marketing department or remote designers. But programmers who experience first-hand other humans having real difficulties tend to respond empathetically and are more motivated to iterate and perfect and polish the overall product for these users—all because their users are real to them.

Deliver the Product

It sounds obvious, but a far too common sin is not shipping the first version soon enough. In fact, some teams never ship "Version 1" but change and hack and perfect the product so long that they ship what can only be called Version 2.

Although sometimes justified by changing market conditions, waiting so long is usually a mistake. By shipping late you may miss a key market window or make your customers unhappy because they had to make do. If you'd shipped when you had a reasonable first version together, you may have released the product before any competitors did.

Support your product once it's shipping. Innovative products in particular need repeated explanation and selling from their designers. Ideally, your talks with users before ever building the product gave you a warm-up on explaining its special benefits.

Listen to Your Market

Despite your fervor in user testing and talking to users, there's no better way to get terrific feedback and guidance on evolving your product than shipping your product and then listening to the market.

Listen to your market by talking to the actual users of your product. Try to find a representative cross-sample of 15 to 25 users who you can talk to in the months after shipping. Question both beginning and advanced users of your product to get their reactions, but be sure to interrogate them again after a few months, when they've had time to plumb all the depths of your product.

Avoid the common mistake of releasing your software too often. Adequate testing and quality assurance at the end of a development project is expensive and often consumes four months. If you release a new version every ten months, you're testing four months for every six of developing. Furthermore, you have hardly any time to incorporate the expert users' feedback from your previous version. It's much better to adopt a 15-18-month cycle of releases, in which you have a full year or so of development for each testing phase. By adopting a discipline of following efficiently spaced releases, you can pull clearly ahead of your competition in features and functionality in just a couple of release cycles.

Reset to Phase One

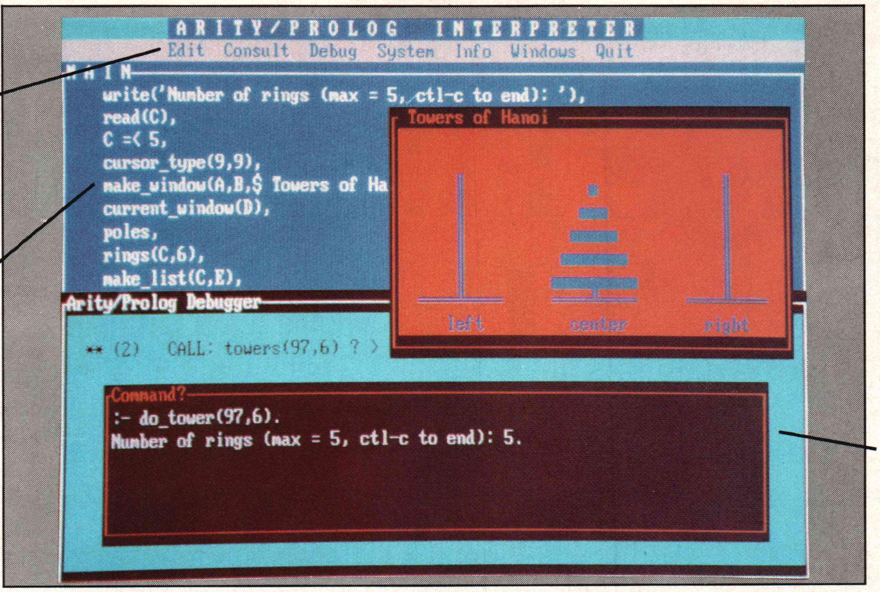
The software delivery cycle is ideally one that loops repeatedly: Each of the eight phases is applied to every major product release, although some, such as selecting tools and building blocks, are obviously much abbreviated on later releases.

Developing for users is not easy. But it provides a methodology that helps you avoid costly errors that are surprisingly common: developing products that nobody wants; or that nobody can learn or use; or that don't meet enough user needs to make them pay money, which is after all what makes us professional developers.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

The right tools make your job easier



Built-in full-screen program editor.

Faster, more efficient interpreter. Full superset of the Edinburgh Prolog standard featuring over 200 predicates designed to meet your programming needs.

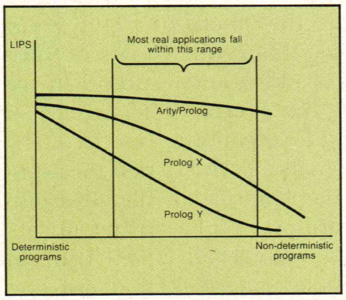
BONUS: Full set of screen design predicates so you can include windows, menus, dialog boxes, and edit boxes like these in your applications. A unique message-passing architecture gives you complete control over the placement, color, and actions of the screen design elements.

Interactive, multiple-window debugger.

Introducing Arity/Prolog Version 5
The right tool for software development

Arity/Prolog Version 5 combines the versatile Prolog language with a fully-integrated user interface and comprehensive language extensions. The result is a fast, full-featured development environment designed to make your job easier.

Speed — when it counts



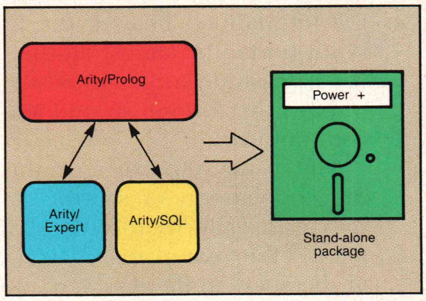
Arity/Prolog V5 sets new standards in Prolog benchmark speeds. Such as 15530 LIPS for Naive Reverse running on an 8 MHz Compaq Portable 286™. But your programs rarely resemble simplistic benchmarks. That's why Arity/Prolog is designed for optimum performance based on typical programming tasks. While the performance of other Prologs declines sharply when tested with real applications, Arity/Prolog's performance remains consistently high.

C, Pascal Language integration — not just interfaces

```
:-public count_it/2:c('_count_it'(int,*int)).

count_it(Init,CountPtr):-
        count(init,Count),
        compute(int=*Countptr,
                *Countptr:=count).

count(100,_):-!,fail.
count(X,Y):-
       inc(X,Y).
count(X,Y):-
       count(X,Y)
```

You don't have to give up familiar tools, like C and Pascal, to take advantage of the power of Arity/Prolog. All C data types and expressions are integrated into Arity/Prolog V5. So you have a choice of using your existing C, Pascal, Fortran, or assembly code with Arity/Prolog V5, or including the declarative programming tasks as part of your Prolog program. And now you can call Arity/Prolog from your C function as well.

Part of a family of application building blocks



Arity also offers **Arity/Expert,** a powerful and flexible expert system development package, and **Arity/SQL,** an ANSI standard implementation of the Structured Query Language for use with the Prolog database. Each of these products is closely integrated with Arity/Prolog V5, making it easy to build customized applications.

That's just the beginning

ARITY

Arity/Prolog V5 includes many more features, such as a virtual database which supports up to 1 gigabyte, database partitioning and indexing, DCG support, and string and floating point support. Arity/Prolog runs on IBM PCs and compatibles. Call today to get more information.

Arity Corporation
30 Domino Drive
Concord, Massachusetts
01742

1-800-PC-ARITY
(Mass: 617-371-1243)

Compaq Portable 286 is a trademark of Compaq Computer Corp.
IBM is a registered trademark of International Business Machines Corp.

Arity/Prolog Compiler and Interpreter ■ Arity/Expert Development Package ■ Arity/SQL Development Package

CIRCLE NO. 85 ON READER SERVICE CARD

Handling Image Files with TIFF

A picture may be worth more than 1,024 words but only if everyone can share the view.

by Anthony Meadow, Rocky Offner, and Michael Budiansky

It may sound categorical, but it's true: Anyone who is writing software that works with bit-mapped images should support the Tag Image File Format (TIFF). Why? Simply because TIFF has become the industry-standard format for storing bit-mapped images; the format is supported by almost all the desktop-publishing and scanner software in both the Macintosh and MS-DOS communities.

TIFF came into existence through the cooperation of several companies, especially Microsoft and Aldus. Many other companies have joined in the effort to develop and support this file format, including most of the scanner manufacturers (such as DEST Corp., Datacopy, and Hewlett-Packard) and the developers of desktop publishing software (Letraset, PS Publishing, and so on). The most recent version is Revision 4, released on April 31, 1987. Revision 5 is now under discussion and adds several important features.

The primary reason for the success of TIFF is the rapid rise of desktop publishing (DTP). DTP software and scanners became so popular that a way was needed to move images from scanners to publishing software. Initially, every scanner manufacturer developed a proprietary format for storing images, but it quickly became obvious that there was a better way to solve this problem. With the support of Aldus and Microsoft, TIFF became the alternative to supporting an odd collection of proprietary file formats. Fortunately, almost all scanner manufacturers decided to support TIFF, which has encouraged almost all developers of desktop-publishing software and paint programs to adopt TIFF, too.

Later in this article, we will describe the TIFF Library Package, which is now in the public domain. The package includes routines for reading and writing TIFF files, a sample application that uses them, and a utility program for examining TIFF files. Also included is a set of TIFF files that most TIFF implementations should be able to read. This set of files was designed to serve as an initial validation suite for TIFF implementations. You can write or call for the full package—the details are at the end of this article.

Anthony Meadow, Rocky Offner, and Michael Budiansky form the developer nucleus of Bear River Associates Inc., P.O. Box 1900, Berkeley, CA 94701.

What Can TIFF Do?

TIFF has satisfied many developers because it is capable of storing all the details about an image, which is not surprising because TIFF was developed as a superset of several existing proprietary formats.

TIFF is unlike most other file formats in that most information is not stored in fixed locations in the file. There are only 8 bytes of information in a TIFF file that have a specified location—the first 8 bytes in the file. Everything else is reached by using offsets from the start of the file. The categories of information that are currently supported seem to be sufficient for almost all applications today. If these categories are not sufficient, then others can be easily added. In fact, even proprietary information can be stored in a TIFF file without violating the specification.

Bit maps of any kind can be stored in a TIFF file—bilevel, grayscale, and color are all supported in Revision 5 of the TIFF standard. Images of any resolution, size, number of samples per pixel, and so on can be stored in TIFF files. These images might come from any kind of device, including scanners, facsimile machines, video frame stores, and so on, or from any kind of software that

can create a bit-mapped image.

TIFF is not supposed to be the only graphics file format. It may not provide sufficient capabilities for all bit-mapped graphics applications, although it's difficult to imagine what the exceptions might be. It does not provide support for object-oriented graphics, whereby images are composed of ellipses, rectangles, splines, and so on (as in Adobe Illustrator, MacDraw, or AutoCAD). TIFF also does not support PostScript or any other page-description language. The PICT and PICT2 formats supported by the Macintosh provide much of the functionality of TIFF with respect to bit maps and much more functionality to support object-oriented graphics and PostScript, but these file formats are proprietary. Table 1, below, compares TIFF with these and other file formats.

The Structure

The TIFF file structure is defined in a specification that is available from Microsoft or with the TIFF Library Package. Revision 4 is the latest, but Revision 5 is now in the works and adds conformance levels, a compression method for color and gray-scale images, and support for a "palette" type of color.

The obvious place to start learning about the TIFF file structure is to read the specification. After that, try dumping several TIFF files with td, the dump utility provided as part of the TIFF Library Package. Look at a variety of files in order to see what you can do with this format. The TIFF Library Package includes about two dozen files that show a wide variety of legal TIFF files.

Abbreviation	Name	Machines on which the format is supported				Support for				
		Mac	MSDOS	Unix	Other	Bilevel Bit Maps	Gray-Scale Bit Maps	Color Bit Maps	O-O Graphics	PostScript
TIFF	Tag Image File Format	Y	Y	Y	Y	Y	Y	Y	N	N
EPS	Encapsulated PostScript	Y	Y	Y	Y	Y[1]	Y[1]	Y[1,2]	Y	Y
PICT	(Macintosh) PICT	Y	N	N	N	Y	N	N	Y	Y[4]
PICT2	(Macintosh) PICT2	Y[3]	N	N	N	Y	Y	Y	Y	Y[4]
RIFF	Raster Image File Format	Y	N	N	N	Y	Y	Y	N	N
MacPaint	(Macintosh) MacPaint	Y	N	N	N	Y[5]	N	N	N	N

O-O graphics means object-oriented graphics (such as MacDraw).

Notes:
1–Bit-maps are supported, but not encouraged. Because EPS files are usually ASCII and not binary in form, bit-maps appear as large runs of digits.
2–Support for color in PostScript will improve when more color printers are available.
3–The PICT2 format is supported only on the Macintosh II at this time.
4–Support for PostScript in PICT and PICT2 files is through the use of special comments, so few applications can do anything with the PostScript commands directly.
5–MacPaint files are always 72 dpi and the maximum document size is 8 × 10 inches.

Table 1: *Comparison of image file formats*

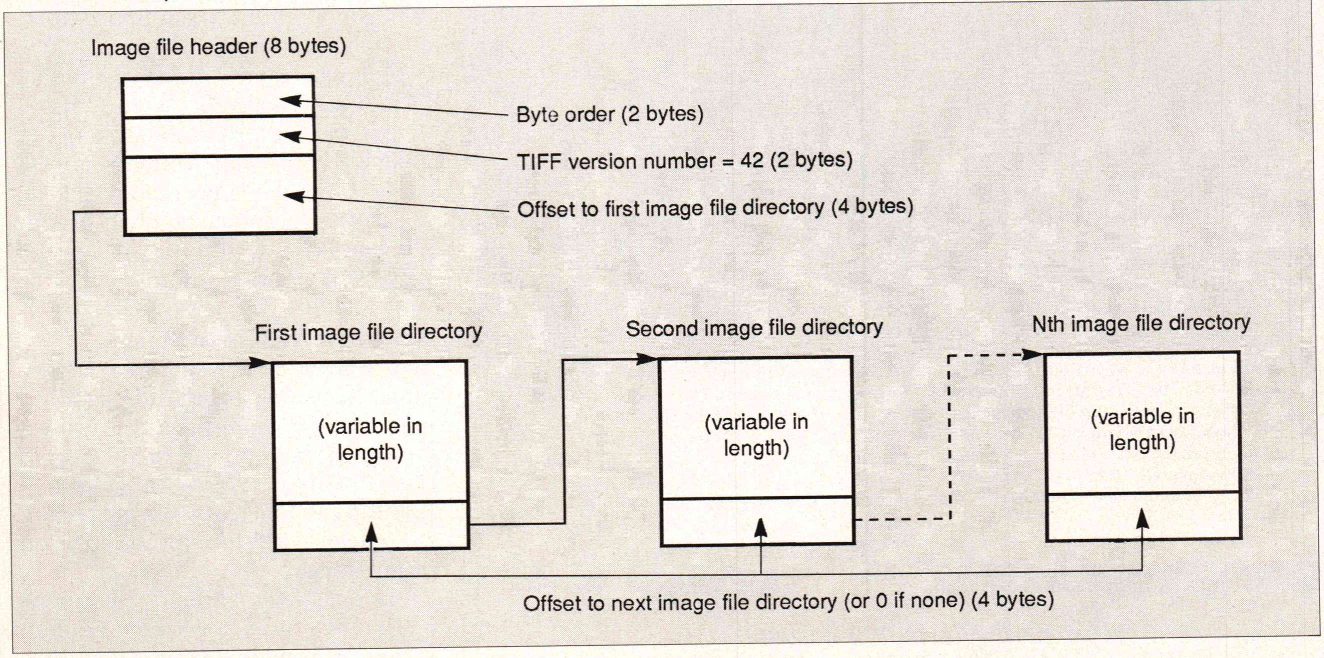


Figure 1: *Structure of a TIFF file*

(continued from page 27)

A TIFF file is composed of three kinds of elements: an image file header, one or more image field directories, and collections of data. Figure 1, page 27, shows the general structure of a TIFF file. The image file header (IFH) always occupies the first 8 bytes in a TIFF file and is the only element that has a fixed position within the file. The image file header contains the byte order flag, the file version number (currently 42), and an offset to the first image file directory (IFD).

All elements other than the IFH have variable positions within the file and are located by using byte offsets from the start of file. A TIFF file is considered to be a continuous sequence of (8-bit) bytes numbered from 0. The largest a TIFF file can be is 232 bytes long, or about 4 Gbytes.

The TIFF specification, like most other specifications, is not especially easy to read. One misunderstanding that we've seen several times concerns the location of various components of the file. The first image file directory is not necessarily just after the image file header—it might be anywhere in the file. Its location is given in the image file header.

A TIFF file may include multiple versions of an image—for example, it might be useful to provide a lower-resolution screen version of a 300-dpi image (to present on a screen), especially if an optimized scaling algorithm was used to generate the lower-resolution version. Of course, an application would have to look for the lower-resolution version and display it rather than generate one on the fly from the 300-dpi version. As far as we know, there are no applications that write more than one version of an image to a file, perhaps because of the additional disk space required. At any rate, if there are multiple versions of an image, then each version is described by the information in its image file directory. Each IFD contains an offset to the next IFD or 0 if there isn't another one.

Even though it is legal to write an image as a single strip, virtually all applications write images as a set of strips

Directories and Tags

Before we look at what an IFD contains, let's look briefly at tags and tag entries. A tag entry is basically a chunk of data (or a field) with a name (or tag). A tag entry might point to the image, the image height, the image width, or the orientation of the image.

An image file directory contains all the tag entries for a version of an image, ordered by tag type. Figure 2, page 32, shows what an IFD looks

The Graphics Toolkit for Contemporary Software Developers

Already the fastest and most powerful graphics toolkit on the market, the new HALO® delivers subroutines and device support for exciting, contemporary applications in publishing, office automation, vision, and image processing.

HALO '88 is a device independent library of 190 graphics subroutines. It is compatible with 18 programming languages, and over 140 hardware devices such as image scanners; graphics, vision, and imaging boards; printers and plotters; and mice. HALO '88 is designed for the complete IBM compatible microcomputer line including the PS/2 and VGA.

Today's Tools for Tomorrow's Applications

HALO '88 has new subroutines which control scanners and scanned images — even images which are larger than screen resolution and available memory. Extended character set support enables software developers to address IBM's full 255 characters in graphics and to design foreign language fonts. Among contemporary HALO '88 applications are CAD, Computer-Based Training, Presentation Graphics, Graphic Arts, Mapping, Machine Vision, Silicon Wafer Manufacturing, Sound System Design, Vehicle Scheduling and Routing, and Real Estate.

Join the HALO Family

HALO has an installed base of 60,000+ end-users, hundreds of site-licensed corporations, government agencies, universities, and national laboratories and most importantly, over 220 Independent Software Developers (ISVs) who market applications written with HALO.

HALO '88 provides the software designer with the richest environment of graphics functions; the programmer with reliable and well-documented tools, and DP managers with continuity of user interface and database format.

HALO™
infinite possibilities in graphics and imaging

Reach for the Future

If you need high performance graphics development software that provides a migration path to OS/2 and other future technology, follow the industry leaders — call (800) 992-HALO (4256).

HALO '88 is just $325 and includes all device drivers, 20 fonts, your choice of one compiler binding, completely new documentation, an interactive tutorial and free 800# technical support. Update from HALO for $150.

Ask about the new HALO Programmers' Workbook which provides C program examples for HALO '88 applications developers.

media cybernetics
8484 Georgia Ave.
Silver Spring, MD 20910
(301) 495-3305, (800) 992-HALO

DESQview API Reference Manual

This is the primary source of information about the DESQview API. It contains all you need to know to write assembly language programs that take full advantage of DESQview's capabilities. The Reference manual comes with an include file containing symbols and macros to aid you in development. AVAILABLE NOW!

DESQview API C Library

The DESQview API C Library provides C Language interfaces for the entire set of API functions. It supports the Lattice C, Metaware C, Microsoft C, and Turbo C compilers for all memory models. Included with the C Library package is a copy of the API Reference Manual and source code for the library. AVAILABLE NOW!

DESQview API Debugger

The DESQview API Debugger is an interactive tool that enables the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter, registers, and stack at the time of the call. Trace conditions can be specified so that only those calls of interest are reported. AVAILABLE JUNE 88

DESQ™ view
Quarterdeck

KEEP UP WITH THE OS/'s

Megabytes of Memory and 32-Bit Performance for DOS.

If you thought the only way to protected mode was the big move to OS/2... We have good news! You can gain the benefits of protected mode the easy way with OS/286™ and OS/386™. These tools for C, Fortran, Pascal and assembly language programmers permit rapid conversion of existing DOS applications from "real" 8086 mode to "protected" 286 and 386 mode. They don't replace or modify DOS, but extend it to protected mode. This way you get multi-megabytes of directly addressable memory (16Mb-286, 4Gb-386) with the compiler, TSRs, device drivers, graphic routines, etc. you use today.

OS/286 and OS/386 are the only DOS extenders that span both the 286 and 386 processors. They run on the widest array of AT and 386 machines, with 32-bit capability *today* on 386s that yields twice the performance of 16-bit mode.

OS/286 and OS/386 are the preferred solutions for developers of high performance memory-intensive applications, including CADKEY, CASE, and Gold Hill, and premier language developers Lahey and MetaWare.

Our optional TOUCHDOWN™ BIOS supplement provides fast and reliable protected mode operation on *any* 286 system, even those with problems resetting the 286. (Ever notice how few existing machines Operating System/2 runs on?) TOUCHDOWN is not required for most major brand AT clones, but for the older machines it is a lifesaver!

If your applications are running out of memory or need more speed, enhance them now without abandoning your investment in DOS.

Coming Soon
WINDOWS 2.0
and DESQview 2.0 API support

Give your protected mode DOS a powerful presentation layer like OS/2, SAA or DESQview.

The HummingBoard® turns any XT, AT or 386 into the fastest, most capable 386 system available. The dual processor architecture provides up to 900K each for multiple real mode applications (Lotus, etc.) which can co-reside with protected mode applications on the HummingBoard's 20MHz 386, while debuggers, editors, networks run concurrently on the base processor. 1-24MB memory, 8,870 Dhrystones. A MUST TRY!

Special $49.50 Evaluation Offer

Check out for yourself the benefits of protected mode. Our $49.50 "sampler special" is a complete OS/286 Developer's Kit, but with a time limited, non-distributable kernel. There's no better way to learn about the outstanding features of OS/286 and OS/386 than to try them. Of course, the $49.50 is applicable to the purchase of the full OS/286 (16-bit) or OS/386 (32-bit) kit at $495 for either one.

The OS/x86 Developer's Kits include support for popular, C, Fortran, Pascal, Lisp, Prolog, compilers, and assemblers from: MetaWare, Lahey, Microsoft, Lattice, Gold Hill, LogicWare and Phar Lap. A number of other packages are also supported including PLINK86, Halo & GSS Graphics, DESQview and soon Windows 2.0.

Run time licenses for OS/286 and OS/386 are available from $40/copy to under $1.00.

A.I. Architects, Inc.

One Kendall Square, Cambridge, MA 02139
TEL (617) 577-8052 FAX (617) 577-9774
Credit card orders only – 24 hours. 617-577-1305

HummingBoard® is a registered trademark and OS/286, OS/386 and TOUCHDOWN are trademarks of A.I. Architects, Inc., PLINK86 is a trademark of Phoenix Corp., HALO is a registered trademark of Media Cybernetics, Inc., DESQview is a trademark of Quarterdeck Office Systems, Windows 2.0 is a trademark of Microsoft Corp.

CIRCLE NO. 79 ON READER SERVICE CARD

(continued from page 28)

like. Only one tag of each tag type is allowed in an IFD.

As shown in Figure 3, below, each tag is 12 bytes long, where the first 2 bytes are the tag type. There are almost 40 tag types, which are defined in the specification. You can use other tags, but you should contact the TIFF administrator at Microsoft first. It's important to reserve any nonspecified tags so that there won't be any conflicts with files created by other applications. The address of the administrator is given at the end of this article.

The next 2 bytes in the tag entry are the data type, and the next 4 bytes are the length (or count). Five data types are defined in the specification: *byte* (1-byte unsigned integer), *ASCII*, *short* (2-byte unsigned integer), *long* (4-byte unsigned integer), and *rational* (two *longs*—the first is the numerator of a fraction and the second is the denominator). The length field in the tag entry gives the length of the data for this tag in terms of the data type—for example, if the data has a length of 1 and is of type *long*, then the data is 4 bytes long. By using the information in these two fields, you know exactly how much data there is for this tag.

The last 4 bytes in the tag entry are either an offset to the data or, if the data occupies 4 bytes or less, the data itself. If the data is less than 4 bytes in size, then it is left-justified within the 4 bytes. This convention for the Offset field optimizes access to small chunks of information, although it does make the file structure more complex.

Stripped Data

Images in TIFF files are usually divided into strips to allow for the memory limitations of most machines. A strip is typically an integral number of scan lines (a scan line is usually a horizontal row). Most applications cannot fit an entire image in memory at one time—for example, an $8^1/_2 \times 11$-inch bilevel image at 300×300 dpi requires more than 1 Mbyte of space. Most Macintoshes don't have this much space available, and most PCs don't have more than 640K of memory to start with.

Even though it is legal to write an image as a single strip, virtually all applications write images as a set of strips, where one strip typically is less than 64K in size. This allows applications to work with images using less memory than would otherwise be needed, although if enough memory is available, an application could still read a complete image into memory. When an image is divided into strips, the strips are either all compressed or all not compressed. The TIFF specification does allow you to compress some strips but not all.

The *StripOffsets* tag entry, as the name implies, contains offsets to the image data and not to the image. Each offset points to one image strip. This is true whether the image is composed of a single strip or many strips. These offsets are the only way to get to the image.

Gray-scale images are stored with all the bits for each pixel packed contiguously. Color images can be stored with all bits per pixel packed together or with the bits describing each color (in the RGB color model) stored in separate planes. The tag *SamplesPerPixel* has a value of 1 for a bilevel or gray-scale image, but for a color picture with three planes, it will have a value of 3. The *PlanarConfiguration* tag tells you whether there is one image plane or several—for example, if a color image was stored as multiple planes, there is one plane for red, a second for green, and a third for blue.

Two other tags always present in TIFF files are *ImageWidth* and *ImageLength*; they contain the width and height of the image in pixels. Each of these is usually 2 bytes long (although a program should always check the Data Type field in the tag entry to find out how long a field really is). For these two tags, the data is actually kept in the Offset field of the tag entry, rather than

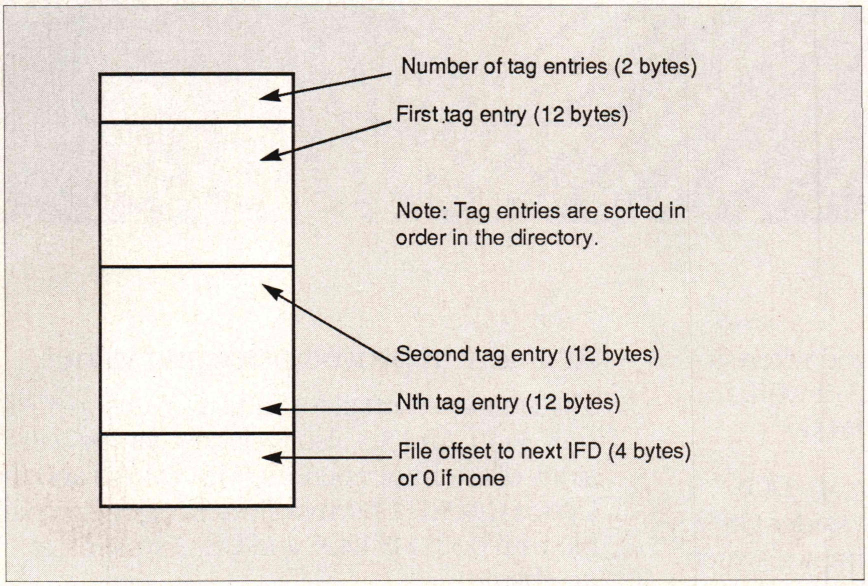


Figure 2: *Structure of an image file directory*

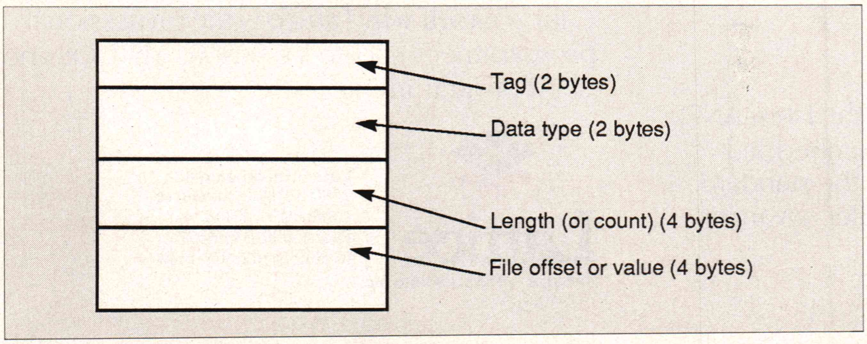


Figure 3: *Structure of a tag entry*

Now the Lattice C Compiler takes you where it's never gone before. With Version 3.3, it works on *two* operating systems: MS-DOS *and* OS/2!

You may now use Version 3.3 on an MS-DOS system to create programs that run under OS/2 protected mode. Or vice versa. A simple "switch" has been put into the compiler to let you generate code for either system or both.

New improved standards...

Version 3.3 is fully compliant with the latest ANSI C standards. It also has improved embedded system support, enhancements to the standard libraries and a host of other compiler advances too numerous to compile.

At a new improved price and value!

The suggested retail for Lattice C Version 3.3 is only $450. And 3.3 also includes "family" versions of the Lattice Screen Editor (LSE) and the Lattice C-SPRITE™ symbolic debugger, compatible with both MS-DOS and OS/2 systems, at no charge.

C for yourself why Lattice is the professional programmer's choice for serious MS-DOS and OS/2 programming.

Lattice
Subsidiary of SAS Institute Inc.

Lattice, Incorporated
2500 S. Highland Avenue
Lombard, IL 60148
Phone: 800/533-3577
In Illinois: 312/916-1600

CIRCLE NO. 142 ON READER SERVICE CARD

TIFF
(continued from page 33)

being pointed to by a file offset.

Compressing Images

Image files are large compared with the average word-processing or spreadsheet document. Now that gray-scale scanners are on the market (and color scanners can't be too far away either), image files could become much larger. Table 2, this page, gives the size of various images at different resolutions. In order to conserve users' disk space, applications should compress images.

Revision 4 of the TIFF standard specifies several compression schemes for bilevel images. Revision 5 adds a method for compression of color and gray-scale. (The compression methods defined in Revision 5 are listed in Table 3, below.) Each of these methods is lossless—that is, each preserves all information in the image. Compression methods are possible that produce much higher compression ratios, but they do not save all of the image's information. These methods could be supported in a TIFF file, but no one has seen the need for them yet. An application needs to support only one compression method when writing each type of image (bilevel, gray-scale, or color) but should be able to read an image in any of the other compression methods.

The default compression method in the specification isn't really compression; it's simply packing data into bytes as tightly as possible. One of the other methods is a variation on this, where the data is packed into (16-bit) words as tightly as possible.

The compression methods for bilevel images are derived from the CCITT (International Telegraph and Telephone Consultative Committee) standards developed for facsimile machines. These methods are based on a Huffman run-length code. The CCITT arrived at the code by looking at samples of typical documents sent by facsimile. It's quite likely that images used in desktop publishing are not like the documents used in developing the CCITT standards, so the compression methods are probably not optimal. They aren't too bad, however; a compression ratio of 4 to 1 is typical. The TIFF standard allows for additional

Resolution* (dpi)	Bits/Pixel 1	4	8	12	24
72	61	242	485	727	1,454
150	263	1,052	2,104	3,156	6,311
300	1,052	4,208	8,415	12,623	25,245
600	4,208	16,830	33,660	50,490	100,980
1,200	16,830	67,320	134,640	201,960	403,920

* Assumes that all images are 8.5 × 11 inches in size
** Assumes that the horizontal and vertical resolution are the same

Table 2: *Sizes of uncompressed bit-mapped images** (K)*

Tag Value	Name
1	None (but pack data into bytes tightly)
2	CCITT Group 3 1-Dimensional modified Huffman run length encoding
3	Facsimile-compatible CCITT Group 3
4	Facsimile-compatible CCITT Group 4
5	Differential run-length encoding
32771	None (same as 1 except pack tightly into words)
32773	PackBits compression

Table 3: *Compression methods in TIFF, Revision 5*

TIFF

(continued from page 35)

compression schemes in the future, but the TIFF file I/O code will have to be revised in all applications so that they can read files that use these new compression methods.

Why Use TIFF?

The most obvious reason to use TIFF is that everyone (well, almost everyone) already does. If you are writing an application that works with bit-mapped images, it can work with all the existing applications that already produce or read TIFF files. If you develop a proprietary file format, you will have to talk many others into supporting it.

The richness of the file structure has caused a couple of other problems.

Another advantage of TIFF is that it was designed to support changes easily. By using new tags, additional information can be added to TIFF files. Older applications won't be able to take advantage of the information in the new tags, but as long as the information they need is there, they can still use the file. Hopefully you won't require any new tags because TIFF already provides a rich set. The kind of information that TIFF supports is a superset of virtually all that contained in proprietary image file formats.

It's also possible to store proprietary information in a TIFF file. The tags numbered 32,768 to 65,535 are reserved for this purpose. Developers who would like one or more tags reserved should contact the TIFF administrator at Microsoft. Obviously, only applications "in the know" are able to use such proprietary information.

Portability of data is becoming more and more important these days. It's now common to see Macs and PCs connected over a local-area network. TIFF supports portability of data because both the Motorola and Intel differences are clearly defined and can therefore be handled easily by an application. TIFF files can also be easily moved to almost any other file system because TIFF makes no assumptions about the underlying file system.

Problems with TIFF

TIFF does have a few problems, but as the standard evolves, many of them are being solved.

The TIFF file structure is not simple—it is more complex than many existing proprietary file formats, such as MacPaint's. This complexity costs time in several ways—for example, there is more overhead to write a TIFF file than a file with a simpler format. It also takes longer to write the TIFF I/O functions for an application because of TIFF's generality, although the library described in this article will reduce this development time for Macintosh developers. There are also TIFF toolkits available from others for MS-DOS machines.

The richness of the file structure has caused a couple of other problems. The TIFF standard (until Revision 5) did not specify a minimal set of tags, so each developer has used a different subset of tags in his or her files. This problem should be solved in Revision 5 of the standard, which specifies six conformance levels. Each level is specified by listing which tags and compression methods must be supported. Hopefully all developers will make the (few) changes in future revisions of their products to bring them to a reasonable conformance level.

Table 4, page 39, outlines the features of the various conformance levels. The FAX conformance level is in a different category from the others because the features required to support facsimile machines are special. An application that supports level 3 and FAX is described as conforming to level 3 + FAX.

Another problem is that there are no compression schemes for color and gray-scale images. An 8.5 × 11-

inch gray-scale image with 8 bits per pixel at 300-dpi resolution requires 8.4 Mbytes of storage without compression. Even people with hard disks would quickly run out of room without some form of compression for these images. This problem is also being solved in Revision 5 of the TIFF standard, which details a compression method for both color and gray-scale.

Tools from the Library

Necessity is the mother of invention. The TIFF Library Package was created as part of DEST Corp.'s product Publish Pac, which is a Macintosh application that lets users operate one of DEST's scanners to read in images and text. (There is also a version of Publish Pac for the IBM PC.) DEST was an early adopter of TIFF; it decided to use TIFF as its standard file format rather than develop yet another proprietary format. The TIFF Library Package is used by Publish Pac to read and write TIFF files. In its current state, it is used in the latest version of Publish Pac. It is written in MPW C (from Apple's Macintosh Programmer's Workshop) and could be ported to other versions of C on the Macintosh with little trouble.

The TIFF Library Package provides low-level routines for working with TIFF files. These routines provide a standard way to read and write TIFF files as well as to manage TIFF tags. The TIFF file format requires a certain amount of bookkeeping, such as ordering all tags sequentially. The library routines handle this automatically to ensure that all images are consistent with the specification. If you use this library, you won't have to learn all the low-level details of what TIFF files look like, but you will still have to decide which tags you want to read and write. The library includes routines to read and write the TIFF header, read and write the tags as a group, and read and write images. Table 5, page 43, contains a list of all the function names.

Reading and writing the file header is straightforward. The two routines *TReadHeader* and *TWriteHeader* read and write the TIFF file header, which contains the byte ordering of the file (Motorola or Intel) and the offset of the first image file directory. If the byte ordering is different from the native ordering, the offset to the directory and all subsequent numerical values are adjusted before they are returned, so an application doesn't have to know whether the file originated on a Macintosh or a PC.

To facilitate tag handling, an in-memory, tag-management scheme was designed so that all tags and their values are read from the file into memory. From there the tags and their values can easily be located, modified, and removed, and new tags can be inserted with simple function calls. When reading a TIFF file, call *TReadTags* to get all the tags into memory at once. You can then make function calls to check for the presence of a tag or get a tag's value.

When you're ready to create a TIFF file, make calls to create or

modify the appropriate tags. The position of the tags and the method for storing the values according to the TIFF specification is handled by the library routines. A call to *TPutPtrTag* or *TPutHdlTag* is made for each tag that's being added to the TIFF file, with two exceptions. The *StripOffsets* and *StripBytesCounts* tags are created and updated automatically by the routine that writes the image strips to the file. Calls to *TWriteTags* include a pointer to the tag's value, and the tag is included in the in-memory tag list.

To write an image to a file, an application makes one call to *TWriteImageStrip* for each strip that will be in the TIFF file. Each strip must have a number of rows less than or equal to that specified by the *RowsPerStrip* tag. Reading an image from a file is a bit more flexible. Any number of rows can be requested by using the *TReadImage* function starting with any row, regardless of the number of rows per strip.

Two auxiliary routines, *TFixOddRowBytes* and *TUnfixOddRowBytes*, provide image adjustment. Using compression method 1, the bits of each row of an image are stored in the smallest possible number of bytes. Some systems normally fit the bits into the smallest number of (16-bit) words—for example, in the Macintosh operating system, bit

Level	Name	Byte Order Supported	No. of Images (IFDs) per File	No. of Tags That Must Be Supported	Bilevel Compression Methods	Gray-Scale Compression Methods	Color Compression Methods
1	Bilevel images with simple compression	Native	1	6	1, 32773	N/A	N/A
2	Rich bilevel images with CCITT compression	Both	1	10	1, 2, 32771, 32773	N/A	N/A
3	Compressed gray-scale images	Both	1	16	1, 2, 32771, 32773	1, 5, 32771	N/A
4	Compressed color images	Both	1	18	1, 2, 32771, 32773	1, 5, 32771	1, 5, 32771
FAX	Bilevel facsimile data	Both	Many	13	1, 2, 3, 4, 32771, 32773	N/A	N/A
5	Unlimited support	Both	Many	37	All defined	All defined	All defined

Table 4: *Conformance levels in TIFF, Revision 5*

We'll Give You Six Solid Reasons Why You Should Be Developing Your Applications In Clarion.

All New Version 2.0

The Clarion Professional Developer Is A Total Programming Environment That Runs On Any IBM PC, PS/2, Or True Compatible With 384K Of Memory And A Hard Disk. The Retail Price Is Just $695. NOT Copy Protected.

CLARION
PROFESSIONAL DEVELOPER™

CIRCLE NO. 103 ON READER SERVICE CARD

1 Slash Your Total Development Effort... From Prototyping To Completion

You'll get live results fast—even before you write the first line of code. Ideas become running applications in a few hours.

Designer, the front-end application generator, allows you to produce major programs with *no coding*. It eliminates prototyping as a preliminary step because design and source code generation are done concurrently. You can implement a dazzling color screen or a report in minutes. You'll probably never code a screen again!

2 Automatically Generate Commented Source Code For Your Entire Application

The Professional Developer's **Designer** is the most powerful application generator in the industry. It creates structured source code that is fully commented. You can easily add to it, modify it, or just admire it.

And The Professional Developer is a complete development environment with all the tools that you need.

3 Create High-Speed, Bullet-Proof Data Management With Built-In LAN Support

Advanced techniques allow you to tune your data management to fit each application. Use related files, data encryption, memo fields, automatic recovery, and commit and rollback—all without compromising Clarion's high level of performance. And complete LAN features are included so the applications you develop will run on your network without any additional run-time or LAN PACK cost.

4 Interface To Your Own C or Assembler Routines For Special Requirements

The Professional Developer will support special device routines and complex logic written in C and Assembler. You won't have to completely re-write all those existing procedures. You can produce completely open-ended solutions that can grow with your needs or requirements.

5 Produce Executable Programs That Don't Need Run-Time Systems

Compile your application into an .EXE program that will run on a stand-alone computer or workstation so you can distribute your programs without costly run-time systems.

6 Get Started Immediately. You'll Be Productive The First Day.

Although there's a lot of horsepower "under the hood," you'll be producing programs soon after you open the package. You'll find the whole environment friendly and comfortable. PC Week says: "Clarion is easy to learn and easy to use."

And One More.

We'll Give You A Free Preview!

See Your Dealer Or Call Toll Free

(800) 354-5444

For A Free Copy Of Our Tutorial Diskette And Introductory Material

(Or, simply return this coupon).

Name ____________________

Company ____________________

Address ____________________

City ____________ State ________ Zip ________

Phone (______) ____________________

Mail This Coupon To:

Clarion Software
150 East Sample Road, Pompano Beach, FL 33064

(continued from page 39)

maps are always an even number of bytes in length. These auxiliary routines are provided to translate the image between these two storage methods if the bits in a row of an image can fit in an odd number of bytes.

The current TIFF Library Package supports most of the first and second conformance levels. Only compression methods 1 (tight packing into bytes) and 2 (modified Huffman run-length encoding) are supported at this time. Also, only the default orientation is supported, where the 0th row represents the top of the image and the 0th column represents the left side of the image (there are seven other possible orientations). When Revision 5 of the TIFF specification is released, the package will probably be upgraded to meet conformance level 3.

The Sample Program

The sample program demonstrates how to use the routines in the Macintosh version of the TIFF Library Package. The program is limited in its ability to display images or read in complex nonbilevel TIFF images, but it does use most of the functions in the library. It can read and write TIFF files and will let you cut and paste Macintosh PICT images to and from the Clipboard—for example, a drawing can be made in MacPaint, cut or copied into the Clipboard, and then pasted into the sample program and written out to a TIFF file. Similarly, a TIFF image can be read into the program, copied into the Clipboard, and pasted into other programs such as MacPaint.

Listing One, page 54, shows two functions—*ReadTiff* and *WriteTiff*—from the sample program. We will now use them to demonstrate the use of the library functions.

The ReadTiff Function

The *ReadTiff* function shows how to use the TIFF library routines to read a TIFF file. First, you read the file header using *TReadHeader*, and then you read all the tags into memory using the *TReadTags* function.

Subsequently, local data structures are filled with the values of the tags via calls to the tag-management routines *TFindTag* and *TGetTag*. If a tag is present, then the value is set; if not and the tag has a default value, the local data for that item is set to the default. If there is no default for a tag, then you either ignore that value or report an error if the tag is necessary.

Once the tag values have been obtained, you determine if there are any values that require facilities beyond those provided to display this image. If so, an error is reported. Once you've determined that the image as described by the tags is correct and that you are able to display that image, a call to *TReadImage* is made. The sample program requests the lines of the image from line 0 through the number of rows in the entire image or the number of rows that will fit into 32K of memory, whichever is smaller. If you successfully read the image, then it (or some portion of it) is displayed in a simple window.

The WriteTiff function

The *WriteTiff* function demonstrates how to write a TIFF file using functions from the TIFF library. A subset of the available tags is placed in the tag list using the *TPutPtrTag* function. The tags specified are those required by the specification plus a couple of others that, although not required by the TIFF specification, are quite useful.

After placing all the tags you want in the tag list, the strips are written

File-handling functions
- TReadHeader
- TWriteHeader
- TReadTags
- TWriteTags
- TReadImage
- TWriteImageStrip

Tag list management functions
- TFindTag
- TGetTag
- TPutPtrTag
- TPutHdlTag

Auxiliary functions
- TFixOddRowBytes
- TUnfixOddRowBytes

Table 5: *TIFF Library Package functions*

SUDDENLY, MAGIC PC MAKES YOUR DBMS OBSOLETE

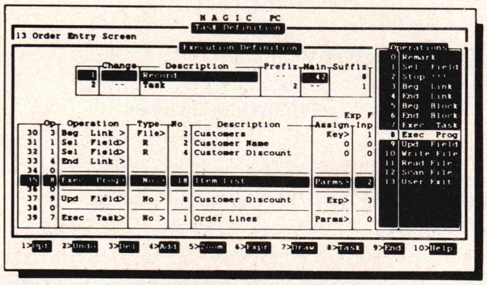
Attn Btrieve™ programmers:

You know how database applications are created — by hacking out line after line of time-consuming code. Most DBMS' and 4GL's give you some programming power. But when it comes to serious applications, they keep you bolted to your seat writing mountains of tedious code. And rewriting it all over again with every design change.

Imagine how much faster you'd be if you could replace the painful coding phase with an innovative visual technology which takes only a fraction of the time: Introducing Magic PC—the revolutionary Visual Database Language from Aker Corporation:

• High-Speed Programming:

With Magic PC's visual design language you quickly describe your programs in non-procedural Execution Tables. They contain compact programming operations which are executed by Magic PC's runtime engine. You fill-in the tables using a visual interface driven by windows and point-and-shoot menus. One table with 50 operations eliminates writing more than 500 traditional lines of code. Yet with Magic PC you don't sacrifice any power or flexibility.



With a powerful set of high-level non-procedural operations you program at only a fraction of the time.

• Maximum Power AND Simplicity:

With Magic PC, you can generate robust DBMS applications including screens, windows, menus, reports, forms, import/export, and much more! Plus, Magic PC has one of the friendliest user interfaces you've ever seen. Using Magic PC you can look-up and transfer data through a powerful Zoom Window system. Magic PC even lets you perform command-free queries.

• Btrieve Performance:

Magic PC incorporates Btrieve, the high-performance file manager from SoftCraft. This gives you exceptional access speed, extended data dictionary capabilities, and automatic file recovery!

• Virtually Maintenance-Free:

With Magic PC you can modify your application design "on the fly" without any manual maintenance. Magic PC automatically updates your programs and data files on-line! This also makes Magic PC an ideal tool for prototyping complete applications in hours instead of days.

• FREE Networking:

Magic PC comes complete with LAN features. Develop multi-user applications for your LAN with Magic's file and record-locking security levels.

• Stand-Along Runtime:

Distribute your applications and protect your design with Magic PC's low cost runtime engine.

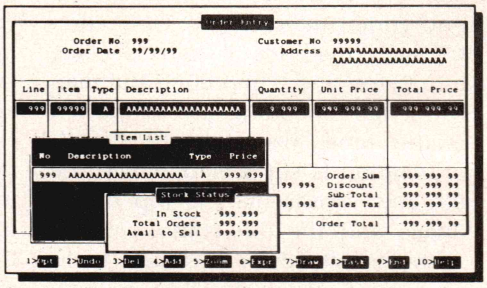
• All For Only $199:

Best of all, Magic PC is an unbeatable bargain. For a limited time, Magic PC's price has been reduced to only $199! Yes, this is the same Magic PC that normally lists for $695! And Magic PC eliminates the need for a separate DBMS, compiler, or application generator. It comes complete with all the tools you need to develop your own database applications instantly.

DBMS Pro's:
MAGIC PC
GIVES YOU MORE
POWER FASTER
BEYOND CODING
~~$695~~
NOW $199.

TECH JOURNAL

"Magic PC's data base engine delivers powerful applications in a fraction of the time. . . there is truly no competitive product."
Victor Wright — PC Tech Journal



Pop-up Zoom Windows run multiple programs per screen — with point-and-shoot data transfer between windows!

• $199 — With A Money-Back Guarantee!

For a limited time, you can get Magic PC for only $199. And even at this low price, Magic PC is risk-free. If you're not completely satisfied, simply return it within 30 days and we'll buy it back (less $19.95 restocking fee). And if you'd like a preview, Magic PC's Tutorial Demo is available for just $19.95.

But you'd better hurry — Magic PC's special $199 price won't last long!

• Join The Magic PC Revolution

To unleash your DBMS design power, order your $199 copy of Magic PC right now by calling toll-free or returning the coupon below.

ORDER NOW: CALL (800) 345-MAGIC
In CA (714) 250-1718

MAGIC PC
The Visual Database Language
by AKER

Yes! I want to generate powerful applications much faster!

☐ Rush me my copy of Magic PC at the special promotional price of $199 (add $10 P&H, and tax in CA. International orders add $30). I understand I can return Magic PC for a refund within 30 days, if I'm not completely satisfied.*

☐ Rush me a copy of Magic PC Tutorial Demo at $19.95 (add $5 P&H, and tax in CA. International orders add $15).

Name ____________________
Company ____________ Phone ____________
Street Address (no POB) ____________________
City ____________ State ______ Zip ______
☐ Check enclosed Charge to my: ☐ VISA ☐ MasterCard ☐ AMERICAN EXPRESS
Account No. ____________________
Acct. Name ____________ Exp. Date ___/___
Signature ____________________
Return to: **Aker Corp.**, 18007 Skypark Cir B2, Irvine, CA 92714

PT DD BS GL DB MS CS CR OP DM AA

System requirements: IBM PC, XT, AT, PS/2 or 100% compatible with 512K RAM, hard disk and DOS 2.0 or later. 5¼" format, not copy protected. Dealer pricing available. *Return policy valid in US only.

TIFF
(continued from page 43)

to the file using the *TWriteImageStrip* function. The number of rows per strip (8) used by the sample program is an arbitrary value chosen to demonstrate how to write multiple strips. The *RowsPerStrip* tag can be set to any value the creator desires, although it is a good idea not to let strips get too big. For the Macintosh, it is wise to limit strips to no more than 32K.

Once all the strips have been written, the tags are flushed to the file by a call to *TWriteTags*. Finally, the header is written using *TWriteHeader*. After the file has been closed and flushed to disk, you've created a TIFF file.

Conclusion

The TIFF file format has become popular in both the Macintosh and MS-DOS worlds. It offers a way to share any kind of bit-mapped image with a large and growing number of other applications. Anyone who is writing software that works with bit maps should support TIFF. The TIFF Library Package described in this article will help any Macintosh programmer read and write TIFF files with a minimum of work. It can also be ported to other machines.

Availability

Because of the size of the TIFF Library Package (close to 1 Mbyte), we've been forced to forego the usual distribution method of printing the listings in the magazine or putting them on CompuServe. Serious developers and readers of *DDJ* can, however, receive a free set of disks for the TIFF Library Package by mentioning this article in a request to one of two sources.

For a copy of the TIFF Library Package for the Macintosh, send your name and address to Bear River Associates Inc., Attn: TIFF, P.O. Box 1900, Berkeley, CA 94701 or call 415-644-9400 (9 A.M. to 5 P.M. PST) and ask for the TIFF Library Package.

A similar TIFF package is available for the PC. To get this package, write to DEST Corp., Attn.: Debra Levesque, 1201 Cadillac Ct., Milpitas, CA 95035 or call 408-946-7100 and ask for Debra Levesque.

To receive a copy of the latest version of the TIFF standard, to reserve a proprietary tag, or to comment on the standard, write to Manny Vellon, Windows Marketing Group, Microsoft Corp., 16011 N.E. 36th Wy., P.O. Box 97017, Redmond, WA 98073-9717.

Bibliography

Andrews, Nancy; and Fry, Stan. "TIFF: An Emerging Standard for Exchanging Digitized Graphics Images." *Microsoft Systems Journal* (July 1987): 71–76.

DDJ

(Listing begins on page 54.)

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

Virtual Arrays in C

The concept of virtual arrays is not new, but it can now be applied to microcomputers.

by Mark Tichenor

This article presents a virtual-array management scheme for C programmers. This simple and flexible approach uses the power of the C language to manipulate arrays that extend themselves automatically and that are limited in size only by the file-size constraints of the operating system. The concept of virtual arrays is not new, but it can now be applied to microcomputers.

Highlights of the scheme are:

- Virtual arrays are stored on disk but are accessed as though they are in memory.
- Automatic file management is provided without explicit reads or writes.
- Disk records can be accessed simply by referencing an array element in any C expression.
- Data can be written to any record in the file simply by using an assignment statement to place a value in an array element.

The Problem

Many problems lend themselves naturally to simple solutions based on the use of data arrays because arrays are easy to manipulate using simple notation. Data arrays can only be as big as available memory allows, however. This is an aggravating limitation because, once arrays outgrow available memory, they are no longer useful.

In implementing an application prototype involving complex tree structures, I chose an array approach using data arrays that incorporated pointers. These pointers were the array indices of other array elements. As the application provided the means to manage a growing volume of information, it was necessary to overcome the memory limitation imposed by the array model.

A lot of thought went into solutions based on dynamic memory allocation of data structures as this approach appeared to promise an elegant and general solution. The need to swap data between memory and disk still remained, however. In fact a whole new problem arose: that of resolving the differences between pointers in memory and pointers on disk.

When dynamically allocated data structures are linked by memory pointers, these pointers become meaningless when written to disk. The linkages must be recreated when the structures are brought back into memory. This problem dramatically increases the complexity of any program that incorporates the swapping of data between memory and disk.

It would be a great advantage to keep using data arrays if their size were not limited by available memory.

The Solution

The memory limitation associated with arrays can be overcome by using virtual arrays that are stored on disk and accessed as though they were in memory. Thus, virtual arrays offer a simple, elegant solution. The C programming language provides the power to manage virtual arrays automatically through the creative use of pointer notation and *#define* macros.

It turns out that virtual-array elements can be accessed by simple reference using a predefined alias and an array index—for example, *item_number(i) = 5;* could be used to set the *item_number* field of the *i*th array element to the value *5*.

Because virtual arrays reside on disk, with paging to memory automatically accomplished behind the scenes, you have a disk-based data management system that operates without any explicit reads or writes. These operations are performed by the virtual-array access function, which is invoked by the predefined access macros.

Mark Tichenor is an instructor in the Business Information Processing Dept., Holland College, Charlottetown Centre, Weymouth St., Charlottetown, PE C1A 4Z1, Canada. He is currently developing interactive software design tools.

The Code

Listing One, page 63, contains the virtual-array header file, and Listing Two, page 63, contains the virtual-array access routines. Four C functions handle all the mechanics of managing the virtual arrays.

The Initialization Routine

```
int init_v_array(filename,
                 record_size, filchar)
char *filename, filchar;
int record_size;
```

Init_v_array creates a new virtual-array file named by filename. File headers are written initializing the record size of the file to *record_size* and setting the number of records to 0. The specified fill character is also placed in the file header for later use in initializing new array elements. The file is then closed. This routine returns a value of 1 if successful, 0 if not.

An example is:

```
if (init_v_array("DATA.VAR",128))
        printf("Success!");
```

which will try to create a new virtual array named *"DATA.VAR"* with elements 128 bytes long and will print a message if successful.

The Open Routine

```
VACB *open_v_array(filename,
                   buffer_size)
char *filename;
int buffer_size;
```

open_v_array prepares an existing virtual array for use. *buffer_size* specifies how many array elements to allocate space for in memory. The routine returns *NULL* if unsuccessful; otherwise, it returns a pointer to the created virtual-array control block (*VACB*).

An example is:

```
VACB *item array;
item_array =
    open_v_array("DATA.VAR",100);
```

which opens the *"DATA.VAR"* array file and reserves enough buffer space for 100 array elements.

The Close Routine

```
void close_v_array(array)
VACB *array;
```

close_v_array writes elements from buffer to disk, closes the array file, and frees allocated memory.

An example is:

```
close_v_array(item_array);
```

The Access Routine

```
void *access_v_array(array,index)
VACB *array;
long index;
```

This routine performs the low-level file management for virtual arrays. It makes sure the array element referenced by *index* is in memory and returns a pointer to it in memory. If the specified array element is already in memory, access is immediate; if not, it is read into memory after saving the record it displaces in the buffer. If the referenced array element does not exist, the routine automatically extends the *array* file so that it does.

For an example, see the listings and the section entitled "Access Notation."

The Access Algorithm

The key to easy reference to array elements (or to fields within them) is the *access_v_rec* function. This function returns a *void* pointer to the location of the element in memory after making sure it is there.

In the interests of demonstrating the feasibility of this approach quickly, I have paid no regard to optimization. The only stipulation was that access to array elements already in memory be as fast as possible.

To meet this requirement, I chose a simple modulus buffering scheme. Each array element has a fixed position in the buffer calculated by element number modulus buffer size. Each element in the buffer is prefixed with a *long* array index containing the number of the element

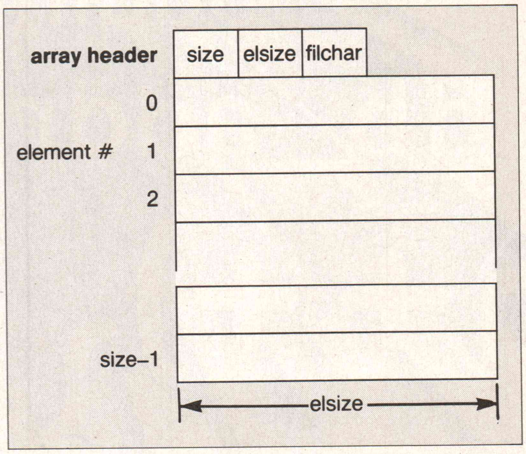


Figure 1: *Virtual-array file layout*

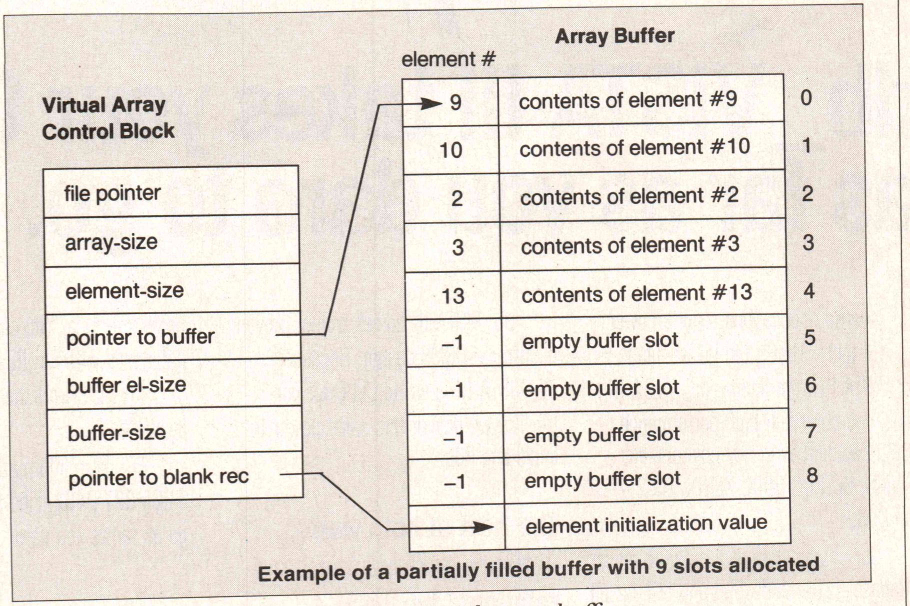


Figure 2: *Virtual-array control block and array buffer*

VIRTUAL ARRAYS
(continued from page 47)

currently present or a –1 if no element is in that buffer position.

If the referenced array index does not match the index of the element currently occupying the calculated buffer location, that array element in the buffer (if any) is written to disk and the referenced element is read in before the buffer address is returned. This scheme avoids searching for records in memory.

Cautions

You should use only *long* integer indices to access array elements. Be sure to *#include* <varray.h>.

Take care not to overrun the end of the array elements when copying strings into them.

You should also take care not to accidentally reference array items far beyond the end of the array unless you really intend to do so. When an array element beyond the end of the array (that is, it does not exist) is referenced, the access routine automatically extends the file so that the referenced element does exist. This process can cause a lot of disk activity and take considerable time because all the elements from the end of the array up to the referenced element must be initialized and written to disk.

Memory copy functions, such as *strcpy*, used to copy data directly from one array element to another are unreliable because the two may occupy the same buffer location. For example:

```
strcpy(desc(n),desc(m));
```

will not work when *n modulus buffer_size* equals *m modulus buffer_size*. However:

```
qty(n) = qty(m) + 1;
```

will work because memory copying is not invoked. (This is true because the compiler calculates the assignment value before it calculates the address for assignment.)

To avoid buffer collision, use:

```
strcpy(temp_desc,desc(m));
strcpy(desc(n),temp_desc);
```

The buffer collision problem associated with the simple modulus buffering scheme could be avoided with the development of a robust least recently used buffering scheme.

Buffered access of any kind also precludes the use of such things as in-memory sort utilities (for example, *qsort*).

Take care not to overrun the end of the array elements when copying strings into them. This would corrupt the index information in the next buffer slot, with unpredictable (probably bad) results.

Access Notation

#define statements are used to simplify access notation both to array elements and to fields within them. One #define is required for each virtual array used. Optionally, one #define may be used to simplify access to each field within a virtual array of structures. Alternately, pointer notation may be used to access fields.

In the example program in Listing Three, page 66:

```
#define VREC(i) ((items *)
      access_v_rec(item_array,i))
```

sets up easy access to elements in the virtual array called item_array (stored in *"ITEMS.VAR"*). In this example, *(items *)* casts the *void* pointer returned by *access_v_rec* to the type *pointer to items*, where *items* is the defined type of the virtual-array element structure. *item_array* is the virtual-array handle returned by *open_v_array*.

In use, *VREC(i)* returns a memory pointer to the *i*th array element wherever it is in the virtual-array buffer. **VREC(i)* is a reference to the *i*th *items* structure in the array. It can

4 TIMES FASTER THAN TODAY'S FASTEST ASSEMBLER!

That's right. 4 times faster.

Clocking in at over 75,000 lines per minute on a 6MHz IBM AT, OPTASM is four times faster than Microsoft's MASM 5.0. **4 times faster — that's 400% more throughput!**

But speed is only one part of it. OPTASM is nearly 100% compatible with MASM 5.0 (except 386 support). It is the only single assembler capable of supporting the various incompatibilities between MASM 3, 4 & 5. That makes OPTASM **more MASM compatible than any single version of MASM!**

Other features? OPTASM generates smaller code without ever generating extra NOP's. It automatically handles jumps out of range, up to 15,000 symbols and most of MASM's phase errors. It also boasts a built in MAKE and simplifies segmentation.

That's why we can make our OPTASM challenge: Test OPTASM head to head against MICROSOFT MASM 5.0. Order **both** assemblers with their 30-day guarantees. In a lot less than 30 days, you'll see just how dazzling OPTASM's speed really is. You'll realize that we're compatible, easier to use, and deliver many more important features than MASM. So accept our challenge. Try both assemblers. **Four times faster and more features, too.** We know which one you'll send back.

Write or call us to order or for our detailed brochure.

OPTASM: $195 Guaranteed returnable within 30 days.

WHAT DO PROGRAMMERS SAY ABOUT OPTASM?

"It (OPTASM) just blows MASM away . . . reduces my assemble time for Periscope from 3-plus minutes to less than 45 seconds."
Brett Salter, President,
The Periscope Company

"OPTASM has been absolutely solid. For me, the most useful new product in 1987."
Chris Dunford, Columbia, MD

SLR SYSTEMS

1622 N. Main Street
Butler, PA 16001
412-282-0864
BBS 412-282-2799
Telex 559215
800-833-3061

CIRCLE NO. 216 ON READER SERVICE CARD

VIRTUAL ARRAYS

(continued from page 49)

be used in any expression as a variable of type *items*.

The expression:

```
#define item(i) VREC(i)->v_item
```

in Listing Three sets up easy access to the *v_item* field in the *i*th element of the *item_array* virtual array. *item(i)* becomes its alias. This allows you to say:

```
item(i) = 24;
```

which is much more intuitive than:

```
((items *)
access_v_rec(item_array,i))->
                    v_item = 24;
```

This ease of reference is what makes this virtual-array system possible.

The size of a virtual array is always available because it is stored in the virtual-array control block. The expression:

```
i = item_array->size;
```

sets *i* to the current size of the virtual array referred to by *item_array*. You should never change this value but can refer to it to determine the index value to use in order to extend the array by one element. For example:

```
i = item_array->size;
item(i) = any_value;
```

will extend the array by one element.

Multidimension arrays are also possible; however, you can extend them only in one dimension. In the example program, the *v_desc* field is a character array that can be regarded as two dimensional when extended over the virtual array. Particular elements in the two-dimensional array can be referenced by *desc(y)[x]*, where *y* is the element number and *x* is the character number.

If you wished to formalize this two-dimensional access, you could add a new macro:

```
#define D(x,y) VREC(y)->v_desc[x]
```

(continued from page 51)

in which *D(x,y)* would refer to column *x*, row *y*. This construct is useful for creating very large virtual-display spaces that can be quickly mapped to the physical display to accomplish fast panning.

Using Virtual Arrays

1. Create a virtual-array file by calling *init__v__array* and passing it the DOS file name, the record length of array elements (in bytes), and a fill character for initializing new elements in the array as they are referenced. This creates a new file, overwriting any existing file of the same name, and writes out the array header information: 4 bytes for array size (initialized to 0), 2 bytes for the size of array elements, and 1 byte for the fill character. Perform this step only once.
2. Typedef your array element structure.
3. Define your array element access macro—for example:

```
#define VREC(i) ((items *)
        access__v__rec(item__array,i))
```

4. Define each field's access macro for the array.
5. Call *open__v__array*, passing it the file name and buffer size. This call must assign the return value to a variable of type *(VACB *)*. This must be the same variable name used in your array access macro—for example:

```
item__array =
    open__v__array("ITEMS.VAR",10);
```

6. Do whatever you want with your virtual array using the access macros you defined in steps 3 and 4.
7. Close your virtual array by calling *close__v__array* and passing it your virtual-array handle—for example, *close__ virtual__array(item__array);*.

Implementation

These routines were tested using Borland's Turbo C development system with both the large- and small-memory models. Using a standard 4.77-MHz PC-compatible with a cheap (slow) hard disk and MS-DOS 3.2, 200 records of 128 bytes each were swapped in 15 seconds and 200 4-byte records were swapped in less than 2 seconds. Though these access speeds may be slow for massive matrix multiplication, they are adequate for many data-access applications.

The beauty of this virtual-array system is that arrays are no longer limited by available memory and all the mechanics of file management take place automatically behind the scenes.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 63.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

The Ada® world just changed. Again.

Meridian continues its tradition of Ada "firsts" by introducing a powerful new set of integrated software development tools.

AdaVantage v2.1 Optimizing Ada Compiler Available for the IBM PC and compatibles and the Apple MacIntosh. An exceptionally fast validated Ada compiler that for the first time brings a true world-class production quality Ada compiler to your desktop personal workstation.

AdaVantage Debugger An interactive source-level debugger for use with programs written using the Meridian AdaVantage compiler. The debugger allows the programmer complete control over the execution of an Ada program in high-level Ada terms — no knowledge of the underlying machine architecture is required. The debugger supports breakpoints, subprogram traces, single-stepping, call backtraces, and full Ada reference syntax.

Ada Developer Interface A powerful interactive screen-oriented interface to the Ada library dependency information. Includes built-in operations to edit, "pretty print", compile, and link any Ada library unit. Configuration file allows user tailoring of all operations and displays.

Run-Time Customization Library For preparation of Ada application programs for execution on 80x86-based embedded systems or other operating systems. The Library is a collection of Ada source files, batch files, and documentation that define the customizable, system-dependent components of the AdaVantage Run-Time System.

AdaStarter Identical to the validated v2.1 AdaVantage compiler with limitations that permit up to ten library units, each with up to two-hundred executable statements (unlimited declarations and comments). This is approximately equivalent to a 4000 line program. The price of AdaStarter is applicable towards purchase of the AdaVantage production compiler. Get the full power of Ada for only $99!

AdaDesigner A collection of tools supporting the design, programming, and documentation phases of the software life cycle. Tools include a structured editor/synthesizer coupled to a text editor/incremental syntax analyzer for Ada, an incremental editor for design languages, a program derivation processor that guarantees your design is always in sync with your implementation, and a structured documentation generator.

Configuration The compilers all run in a standard PC configuration with 640K of memory (1MB on the MacIntosh) and a hard disk.

For more information call 1-800-221-2522.
In California, call 714-380-9800.

"Meridian Software Systems AdaVantage version 2.0 compiler is an Ada software milestone." — PC Week

MERIDIAN
SOFTWARE
SYSTEMS, INC.

23141 VERDUGO DRIVE • SUITE 105 • LAGUNA HILLS, CALIFORNIA 92653
800/221-2522 (Outside California) • 714/380-9800 (Inside California)
Telex: 650-268-0547 MCI • Fax: 714/380-1683

CIRCLE NO. 154 ON READER SERVICE CARD

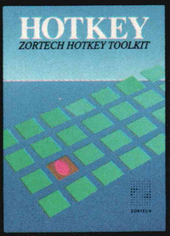
Listing One *(Text begins on page 26.)*

```
/* Primary Interface Files */
#include "Types.h"
#include "Quickdraw.h"
#include "Windows.h"

/* Other Interface files */
#include "Errors.h"
#include "Files.h"
#include "Memory.h"
#include "Packages.h"
#include "Scrap.h"

/* Application-specific Include files */
#include "::TiffLibrary:TIFFLib.h"
#include "sample.h"
#include "messages.h"

static Ptr  SetIDPtr();
static void CleanUp();
static void InitID();

#define     MAXIMAGESIZE    0x8000                          /* Limit images to 32K for now */
#define     INFINITY                        0x4000000       /* TIFF Spec says 2**32-1 but this is big enough */

/* Read in an image from a TIFF format file.  As the code demonstrates, we
 * do not read in very complicated images.  We read in a number of tags,
 * and reject the image as an unsuitable tiff file if any of several
 * conditions exist.  We do not read in images that have more than one bit
 * of image data per pixel.  Of those simple images that we do read, we
 * will only read the first 32k of that image.
 */
Boolean ReadTiff(refNum, myBitMapPtr)
Int16 refNum;
BitMap *myBitMapPtr;
{
            Handle                                          listH;
            Boolean                                         oddRowBytes;
            Int8                                            dummy;
            Int16                                           byteOrder,
                                                                    rowBytes,
                                                                    scrnHRes,
                                                                    scrnVRes;
            Int32                                           tagOffset,
                                                                    nextDirOffset,
                                                                    nextFileFree,   /* next free location in output file */
                                                                    dirOffset,
                                                                    rowsPerImage,
                                                                    count,
                                                                    size;
            Rational                        xRes,
                                                                    yRes;
            Rect                                            imageRect;
            TiffDirEntry    tagDirEntry;
            id                                                      id;                     /* image
description */

            InitID(&id);
            ScreenRes(&scrnHRes, &scrnVRes);                /* if needed for defaults */

            /*
             * Read in header and Tags
             */

            if (TReadHeader(refNum, &dirOffset, &byteOrder) != noErr) {
                        ErrorMessage(BADREADHEADER);
                        return(false);
            }

            if(TReadTags(refNum, byteOrder,
                                                            &listH, dirOffset, &nextDirOffset) != noErr) {
                        ErrorMessage(BADREADTAGS);
                        return(false);
            }

            /* Get tags values.
             */
            /* SUBFILE_TYPE_TAG */
            if (TFindTag(listH, &tagOffset, SUBFILE_TYPE_TAG))
                        TGetTag(listH, tagOffset, &id.subfileType, sizeof(id.subfileType));
            else {
                        ErrorMessage(BADTIFF);
                        CleanUp(listH, &id);
                        return(false);
            }
            /* IMAGE_WIDTH_TAG */
            if (TFindTag(listH, &tagOffset, IMAGE_WIDTH_TAG))
                        TGetTag(listH, tagOffset, &id.imageWidth, sizeof(id.imageWidth));
            else {
                        ErrorMessage(BADTIFF);
                        CleanUp(listH, &id);
                        return(false);
```

(continued on page 56)

NEW! TOOLKITS FOR TURBO C & QUICK C from ZORTECH INC.

ONLY $49.95 EACH



HOTKEY

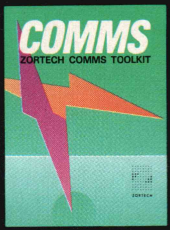
A complete set of Terminate Stay Resident (TSR) functions that help you to write reliable 'pop-up' programs.

Now you can make your programs 'Sidekickable'. Two example programs are included, a 'pop-up Calculator' and a pop-up 'Critical Error Handler'.

The Hotkey toolkit handles all floating point functions in resident mode.

The 32 page manual includes an interesting discussion of the origin and history of undocumented MS-DOS function calls, together with a full explanation of the theory and practical use of TSR's.

Only $49.95! (State Turbo C or Quick C version.)



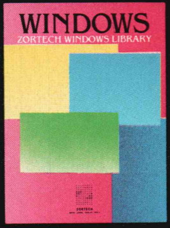
COMMS

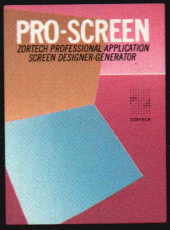
Do you need to incorporate serial communications into your applications? Yes! Then get this inexpensive but highly professional COMMS toolkit from Zortech Inc.

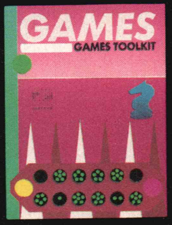
Look at the list of features: Xmodem, Kermit and ASCII file transfer, Hayes modem control, VT52, VT100 and ANSI terminal emulation, supports up to 8 serial ports, speeds up to 19.2k baud rate and higher.

Two demonstration programs are included, MINICOM and MAXICOM (like Procomm) together with the 120 page manual and full source code FREE!

Only $49.95! (State Turbo C or Quick C version.)







GAMES

Have you ever wondered how to write a chess program? Now we reveal the secret algorithms and techniques of the masters with this dynamic Games toolkit.

The package comes complete with the full source code to three ready to play games of strategy – Chess, Backgammon and Wari (an ancient African game).

A comprehensive 150 page manual is provided giving an in depth look at the history, structure and program design of such 'Strategy Games'.

Only $49.95!

(State Turbo C or Quick C version.)

SUPERTEXT

This is not simply an 'Editor' toolkit, but a full-blown, 'WordStar' compatible wordprocessor with the full source code.

As well as all the normal editing functions, you will also find 'dot' commands and full printer control. The SuperText toolkit handles files of any size and allows full on-screen configuration.

Do you need to incorporate a wordprocessor into your application? Yes! Then get the SuperText toolkit complete with full source code and 150 page manual now!

Only $49.95! (State Turbo C or Quick C version.)

PROSCREEN

Generate high quality data entry screens with the Pro-Screen – Screen Designer and Code Generator.

You can draw the data entry screen, define the input fields, define the input criteria, set screen colors and attributes, draw single or double lines, make boxes – press a few buttons and 'hey presto' Pro-Screen generates the C source code for your application!

Professional applications programmers will find this versatile utility and it's associated functions invaluable.

Comes complete with a substantial 78 page manual and demo programs.

Only $49.95! (State Turbo C or Quick C version.)

WINDOWS

Add super-fast text screen handling to your applications with the WINDOWS library from Zortech Inc.

Give your applications the professional look – with instant zooming and exploding windows. Incorporate drop-down menus and Lotus style menus with our easy to use functions.

Automatically handles memory saving and buffering of window text. Use any number of overlapping windows in your applications. Write to any window, read from any window, close any window, pull any window to the top.

Over 55 functions together with a big 85 page manual and remember, you get the full source code.

Only $49.95! (State Turbo C or Quick C version.)

NEW! C VIDEO

- Now learn C the easy way!
- Get the 'Complete C Video Course' from Zortech Inc. together with our big 365 page workbook.
- Ten 1 hour tapes – 36 lessons!
- Easy to follow course, you get an excellent introduction to the C language.
- Takes you step-by-step up to the intermediate and advanced levels.
- Teach yourself at home or the office – at your own speed.

only $295.00!

Yes! Rush me these items!

☐ HOTKEY
☐ COMMS
☐ PRO-SCREEN
☐ WINDOWS ☐ GAMES
☐ SUPERTEXT ☐ C VIDEO

FREE SHIPPING – VISA/MC/COD/CHECK

Name ....

Address ....

.... Phone ....

VISA or MC# .... Exp. Date ....

ZORTECH Inc. 361 Massachusetts Ave, Arlington, MA 02174

Support & Enquiries Tel: (617) 646-6703

ORDER HOTLINE (800) 848-8408

ZORTECH
BOSTON LONDON FRANKFURT GENEVA

Listing One *(Listing continued, text begins on page 26.)*

```
        }
        /* IMAGE_LENGTH_TAG */
        if (TFindTag(listH, &tagOffset, IMAGE_LENGTH_TAG))
                TGetTag(listH, tagOffset, &id.imageLength, sizeof(id.imageLength));
        else {
                ErrorMessage(BADTIFF);
                CleanUp(listH, &id);
                return(false);
        }
        /* ROWS_PER_STRIP_TAG */
        if (TFindTag(listH, &tagOffset, ROWS_PER_STRIP_TAG)) {
                TGetTag(listH, tagOffset, &id.rowsPerStrip, sizeof(id.rowsPerStrip));
                tagDirEntry = GetDirEntry(listH, tagOffset);
                switch(tagDirEntry.type) {
                case LONG:
                                break;
                case SHORT:     /* ok, but convert returned value */
                                id.rowsPerStrip = (long)( *((Int16 *)(&id.rowsPerStrip)) );
                                break;
                default:
                                ErrorMessage(BADTIFF);
                                CleanUp(listH, &id);
                                return(false);
                }
        }
        else {
                id.rowsPerStrip = INFINITY;
        }
        /* SAMPLES_PER_PIXEL_TAG */
        if (TFindTag(listH, &tagOffset, SAMPLES_PER_PIXEL_TAG))
                TGetTag(listH, tagOffset, &id.samplesPerPixel, sizeof(id.samplesPerPixel));
        else {
                id.samplesPerPixel = 1;
        }
        /* BITS_PER_SAMPLE_TAG */
        id.bitsPerSample = SetIDPtr(listH, BITS_PER_SAMPLE_TAG, 1L, SHORT);
        if (id.bitsPerSample == nil) {
                                CleanUp(listH, &id);
                                return(false);
        }
        /* PLANAR_CONFIG_TAG */
        if (TFindTag(listH, &tagOffset, PLANAR_CONFIG_TAG))
                TGetTag(listH, tagOffset, &id.planarConfig, sizeof(id.planarConfig));
        else
                id.planarConfig = 1;
        /* COMPRESSION_TAG */
        id.compression = SetIDPtr(listH, COMPRESSION_TAG, 1L, SHORT);
        if (id.compression == nil) {
                                CleanUp(listH, &id);
                                return(false);
        }
        /* MIN_SAMPLE_VALUE_TAG */
        id.minSampleValue = SetIDPtr(listH, MIN_SAMPLE_VALUE_TAG, 0L, SHORT);
        if (id.minSampleValue == nil) {
                                CleanUp(listH, &id);
                                return(false);
        }
        /* MAX_SAMPLE_VALUE_TAG */
        id.maxSampleValue = SetIDPtr(listH, MAX_SAMPLE_VALUE_TAG,
                                                                        (Int32)((1 <<
    *id.bitsPerSample) - 1), SHORT);
        if (id.maxSampleValue == nil) {
                                CleanUp(listH, &id);
                                return(false);
        }
        /* PHOTOMETRIC_INTERP_TAG */
        if (TFindTag(listH, &tagOffset, PHOTOMETRIC_INTERP_TAG))
                TGetTag(listH, tagOffset, &id.photoInterp, sizeof(id.photoInterp));
        else {
                id.photoInterp = 0;             /* assume mac photometric interpretation */
        }
        /* FILL_ORDER_TAG */
        if (TFindTag(listH, &tagOffset, FILL_ORDER_TAG))
                TGetTag(listH, tagOffset, &id.fillOrder, sizeof(id.fillOrder));
        else
                id.fillOrder = 1;
        /* ORIENTATION_TAG */
        if (TFindTag(listH, &tagOffset, ORIENTATION_TAG))
                TGetTag(listH, tagOffset, &id.orientation, sizeof(id.orientation));
        else
                id.orientation = 1;
        /* X_RESOLUTION_TAG */
        if (TFindTag(listH, &tagOffset, X_RESOLUTION_TAG))
                TGetTag(listH, tagOffset, &id.xResolution, sizeof(id.xResolution));
        else {
                id.xResolution.numerator = (Int32)scrnHRes;
                id.xResolution.denominator = 1;
        }
        /* Y_RESOLUTION_TAG */
        if (TFindTag(listH, &tagOffset, Y_RESOLUTION_TAG))
                TGetTag(listH, tagOffset, &id.yResolution, sizeof(id.yResolution));
```

(continued on page 59)

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

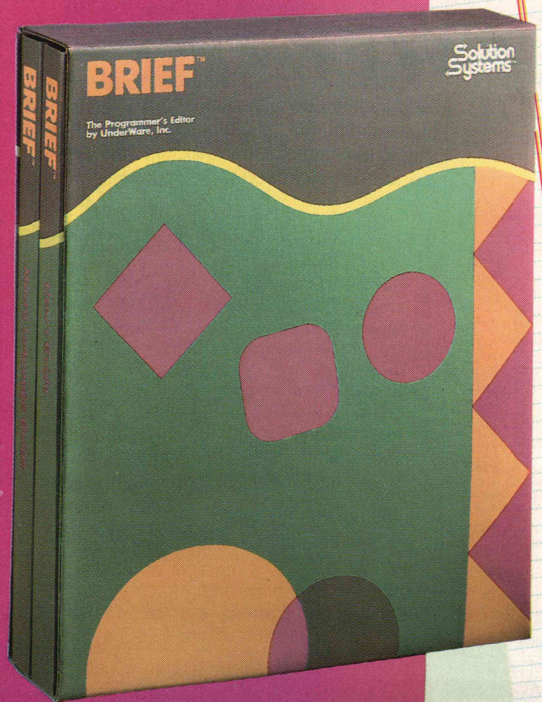
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492
(in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410D
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.
BRIEF is a trademark of UnderWare, Inc.
Solution Systems is a trademark of Solution Systems.

Listing One *(Listing continued, text begins on page 26.)*

```
        else {
                    id.yResolution.numerator = (Int32)scrnVRes;
                    id.yResolution.denominator = 1;
        }

        /* Initialize of non-tag values.
         */
        oddRowBytes = (((id.imageWidth * (*id.bitsPerSample)) + 7) / 8) % 2 != 0;
        id.stripsPerImage =
                                        (id.imageLength + id.rowsPerStrip - 1) / id.rowsPerStrip;

        /*          Check Tag Values to see if we can read this TIFF file.
         *
         *          NOTE: Although the majority of the tag were read, all the values
         *          obtained are not used in displaying the image in THIS PROGRAM.
         *          Those not used were read in simply to provide the example.
         */
        if ( (id.samplesPerPixel != 1)                                              ||
                     (*id.bitsPerSample != 1)                                        ||
                     (id.planarConfig != 1 && id.planarConfig != 2) )          {
                    ErrorMessage(BADTIFF);
                    CleanUp(listH, &id);
                    return(false);
        }
        if (id.photoInterp != 0)        /* We don't translate yet so let 'em know */
                    ErrorMessage(BADPHOTOINTERP);

        rowBytes = (((id.imageWidth - 1) / (2 * 8)) + 1) * 2;
        /* only make image as much as will fit in MAXIMAGESIZE for now */
        size = id.imageLength * rowBytes;
        if (size > MAXIMAGESIZE) {
                    ErrorMessage(IMAGECROPWARN);
                    size = MAXIMAGESIZE;
        }
        rowsPerImage = size / rowBytes;

        /* Prepare bitmap.
         */
        if (myBitMapPtr->baseAddr != nil)
                    DisposPtr(myBitMapPtr->baseAddr);
        if ((myBitMapPtr->baseAddr = MyNewPtr(size)) == nil) {
                    CleanUp(listH, &id);
                    return(false);
        }
        myBitMapPtr->rowBytes = rowBytes;
        myBitMapPtr->bounds.top = 0;
        myBitMapPtr->bounds.left = 0;
        myBitMapPtr->bounds.bottom = rowsPerImage;
        myBitMapPtr->bounds.right = id.imageWidth;


        /* Read in image.
         */
        if (TReadImage(refNum, listH,
                                        0L, myBitMapPtr->baseAddr, rowsPerImage, -1) != noErr) {
                    CleanUp(listH, &id);
                    return(false);
        }
        if (oddRowBytes)
                    TFixOddRowBytes(myBitMapPtr);
        DisplayImage(FrontWindow(), myBitMapPtr);

        CleanUp(listH, &id);
        return(true);
}

void WriteTiff(refNum, myBitMapPtr)
Int16 refNum;
BitMap *myBitMapPtr;
{
        Handle                  listH;
        Ptr                                 bufferPtr;
        Boolean                 oddRowBytes;
        Int8                    dummy;
        Int16                   byteOrder,
                                            subfileType,
                                            imageWidth,
                                            imageLength,
                                            fillOrder,
                                            compressType,
                                            photoInterp,
                                            bitsPerPixel,
                                            minSampleValue,
                                            maxSampleValue,
                                            orientation,
                                            tiffRowBytes,   /* number of bytes per row in TIFF format */
                                            plane,                          /* dummy parameter for
TWriteImageStrip */
                                            scrnHRes,
                                            scrnVRes;
        Int32                   rowsPerStrip,
```

(continued on next page)

Listing One *(Listing continued, text begins on page 26.)*

```
                                                            nextFileFree,    /* next free location in output file */
                                                            startLine,
                                                            numLines,
                                                            dirOffset,
                                                            count;
          Rational        xRes,
                                                            yRes;
          Rect                          imageRect;

          /* get a handle for the in memory tag list */
          listH = NewHandle(0);
          if (MemError() != noErr) {
                          ErrorMessage(BADMEMORY);
                          return;
          }

          ScreenRes(&scrnHRes, &scrnVRes);
          imageRect = myBitMapPtr->bounds;
          /* write out 8 rows per strip - 8 is an arbitrary number */
          rowsPerStrip = MIN(imageRect.bottom - imageRect.top, 8);

          /* initialize tag values */
          byteOrder = MOTOROLA;
          subfileType = 1;
          imageWidth = imageRect.right;
          imageLength = imageRect.bottom;
          bitsPerPixel = 1;
          fillOrder = 1;
          compressType = 1;
          photoInterp = 0;
          minSampleValue = 0;
          maxSampleValue = (1 << bitsPerPixel) - 1;
          orientation = 1;
          xRes.numerator = (Int32)scrnHRes;
          xRes.denominator = 1;
          yRes.numerator = (Int32)scrnVRes;
          yRes.denominator = 1;

          tiffRowBytes = (imageRect.right * bitsPerPixel + 7) / 8;
          oddRowBytes = (tiffRowBytes % 2) != 0;

          /* Put tags in memory list.
           * The order tags are put in the list with TPutPtrTag is NOT important.
           */
          if (            TPutPtrTag(listH, SUBFILE_TYPE_TAG, SHORT,
                                                                                                      1L,
&subfileType)             != noErr        ||
                                          TPutPtrTag(listH, IMAGE_WIDTH_TAG,              SHORT,
                                                                                                      1L,
&imageWidth)              != noErr        ||
                                          TPutPtrTag(listH, IMAGE_LENGTH_TAG, SHORT,
                                                                                                      1L,
&imageLength)             != noErr        ||
                                          TPutPtrTag(listH, ROWS_PER_STRIP_TAG, LONG,
                                                                                                      1L,
&rowsPerStrip)            != noErr        ||
                                          TPutPtrTag(listH, X_RESOLUTION_TAG, RATIONAL,
                                                                                                      1L, &xRes)
          != noErr        ||
                                          TPutPtrTag(listH, Y_RESOLUTION_TAG, RATIONAL,
                                                                                                      1L, &yRes)
          != noErr        ||
                                          TPutPtrTag(listH, BITS_PER_SAMPLE_TAG, SHORT,
1L,&bitsPerPixel)         != noErr        ||
                                          TPutPtrTag(listH, COMPRESSION_TAG, SHORT,
                                                                                                      1L,
&compressType)            != noErr        ||
                                          TPutPtrTag(listH, FILL_ORDER_TAG, SHORT,
                                                                                                      1L,
&fillOrder)               != noErr        ||
                                          TPutPtrTag(listH, ORIENTATION_TAG, SHORT,
                                                                                                      1L,
&orientation)             != noErr        ||
                                          TPutPtrTag(listH, PHOTOMETRIC_INTERP_TAG, SHORT,
                                                                                                      1L,
&photoInterp)             != noErr        ) {
                          ErrorMessage(BADPUTTAGS);
                          return;
          }

          /* leave room for header in output file */
          SetEOF(refNum, (Int32)sizeof(TiffHeader));
          nextFileFree = sizeof(TiffHeader);

          /* Write out image to file, fixing from Macintosh rounding of rows to the
           * nearest 2 bytes, to the TIFF rounding of rows to the nearest byte.
           */
          if (oddRowBytes)
                          TUnfixOddRowBytes(myBitMapPtr);              /* round rows to nearest byte */
```

```
          startLine = 0;
          bufferPtr = myBitMapPtr->baseAddr;
          while  (startLine < imageLength) {
                    numLines = MIN(imageLength - startLine, rowsPerStrip);
                    if (TWriteImageStrip(refNum, &nextFileFree, listH,
                                                                              startLine, numLines, bufferPtr, plane) !=
noErr) {
                              ErrorMessage(BADWRITEIMAGE);
                              return;
                    }
                    startLine += numLines;
                    bufferPtr += numLines * tiffRowBytes;
          }

          if (oddRowBytes)
                    TFixOddRowBytes(myBitMapPtr);                /* round rows to nearest word */

          /* directory must be on word boundary */
          if (nextFileFree % 2 != 0) {
                    /* add filler byte */
                    count = 1;
                    if (FSWrite(refNum, &count, &dummy) != noErr) {
                              ErrorMessage(BADWRITE);
                              return;
                    }
                    nextFileFree++;
          }

          dirOffset = nextFileFree;

          if (TWriteTags(refNum, byteOrder, &nextFileFree, listH, 0L) != noErr) {
                    ErrorMessage(BADWRITETAGS);
                    return;
          }
          if (TWriteHeader(refNum, dirOffset, byteOrder) != noErr) {
                    ErrorMessage(BADWRITEHEADER);
          }
}

/* Free the list handle and any memory allocated to image description structure.
 */
static void CleanUp(listHandle, idPtr)
Handle listHandle;
id *idPtr;
{
          MyDisposPtr(&idPtr->bitsPerSample);
          MyDisposPtr(&idPtr->compression);
          MyDisposPtr(&idPtr->docName);
          MyDisposPtr(&idPtr->imageDescription);
          MyDisposPtr(&idPtr->make);
          MyDisposPtr(&idPtr->model);
          MyDisposPtr(&idPtr->stripOffsets);
          MyDisposPtr(&idPtr->stripByteCounts);
          MyDisposPtr(&idPtr->minSampleValue);
          MyDisposPtr(&idPtr->maxSampleValue);
          MyDisposPtr(&idPtr->pageName);
          MyDisposPtr(&idPtr->freeOffsets);
          MyDisposPtr(&idPtr->freeByteCounts);
          MyDisposPtr(&idPtr->grayResponseCurve);
          MyDisposPtr(&idPtr->colorResponseCurves);
          DisposHandle(listHandle);
}

void InitID(idPtr)
id *idPtr;
{
          idPtr->subfileType = -1;
          idPtr->imageWidth = 0;
          idPtr->imageLength = 0;
          idPtr->bitsPerSample = nil;
          idPtr->compression = nil;
          idPtr->photoInterp = -1;
          idPtr->threshholding = -1;
          idPtr->cellWidth = -1;
          idPtr->cellLength = -1;
          idPtr->fillOrder = 0;
          idPtr->docName = nil;
          idPtr->imageDescription = nil;
          idPtr->make = nil;
          idPtr->model = nil;
          idPtr->stripOffsets = nil;
          idPtr->orientation = -1;
          idPtr->samplesPerPixel = 0;
          idPtr->rowsPerStrip = 0;
          idPtr->stripsPerImage = 0;
          idPtr->stripByteCounts = nil;
          idPtr->minSampleValue = nil;
          idPtr->maxSampleValue = nil;
          idPtr->xResolution.numerator = 0;
          idPtr->xResolution.denominator = 0;
          idPtr->yResolution.numerator = 0;
          idPtr->yResolution.denominator = 0;
          idPtr->planarConfig = -1;
          idPtr->pageName = nil;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 26.)*

```
		idPtr->xPosition.numerator = 0;
		idPtr->xPosition.denominator = 0;
		idPtr->yPosition.numerator = 0;
		idPtr->yPosition.denominator = 0;
		idPtr->freeOffsets = nil;
		idPtr->freeByteCounts = nil;
		idPtr->grayResponseUnit = -1;
		idPtr->grayResponseCurve = nil;
		idPtr->group3Options = 0;
		idPtr->group4Options = 0;
		idPtr->resolutionUnit = -1;
		idPtr->pageNumber[0] = 0;
		idPtr->pageNumber[1] = 0;
		idPtr->colorResponseUnit = -1;
		idPtr->colorResponseCurves = nil;
}

TiffDirEntry GetDirEntry(listHandle, tagOffset)
Handle		listHandle;
Int32		tagOffset;
{
		Ptr						p;

		/* get pointer to tag list */
		p = &(**listHandle);		/* HANDLE DEREFERENCE */
		/* get pointer to our tag's directory entry */
		p += tagOffset;
		/* return the whole Directory Entry Structure, not a pointer to it */
		return(*((TiffDirEntry *)p));
}


Int16 TypeSize(type)
Int16 type;
{
		switch (type) {
			case BYTE:
				return(BYTESIZE);
			case ASCII:
				return(ASCIISIZE);
			case SHORT:
				return(SHORTSIZE);
			case LONG:
				return(LONGSIZE);
			case RATSIZE:
				return(RATSIZE);
		}
}

Ptr SetIDPtr(listHandle, tag, defaultValue, defaultType)
Handle listHandle;
Int16 tag;
Int32 defaultValue;			/* won't handle defaults larger than 4 bytes */
Int32 defaultType;
{
		TiffDirEntry	tagDE;
		Ptr						p;
		Int32				tagOffset,
								size,
								nvals;

		if (TFindTag(listHandle, &tagOffset, tag)) {
			tagDE = GetDirEntry(listHandle, tagOffset);
			size = TypeSize(tagDE.type) * tagDE.length;
			if ((p = MyNewPtr(size)) != nil)
				TGetTag(listHandle, tagOffset, p, size);
		}
		else if (defaultType != 0) {
			if ((p = MyNewPtr(defaultType)) != nil) {
				switch (defaultType) {
					case BYTE:
						*(unsigned char *)p = defaultValue;
						break;
					case ASCII:
						*(unsigned char *)p = defaultValue;
						break;
					case SHORT:
						*(unsigned short *)p = defaultValue;
						break;
					case LONG:
						*(unsigned long *)p = defaultValue;
						break;
					case RATIONAL:
						((Rational *)p)->numerator = defaultValue;
						((Rational *)p)->denominator = 1;
						break;
				}

				}
			}
			else
				p = nil;
			return(p);
	}
```

End Listings

Listing One *(Text begins on page 46.)*

```
Listing One

/* Virtual Array Header File */
#include <alloc.h>
#include <stdio.h>

/* Virtual Array Control Block typedef */

typedef struct {

   FILE *file;            /* pointer to file control block */
   long size;             /* number of array elements in file */
   int  elsize;           /* number of bytes in each element */
   char *buffer;          /* pointer to array buffer */
   int  buf_elsize;       /* size of element in buffer including index */
   int  buf_size;         /* number of elements in buffer */
   char *blank_rec;       /* pointer to initialization record */
                          /* used for extending file */
} VACB ;                  /* virtual array control block type name */

/* Virtual Array Access Prototypes */

int init_v_array(char *filename,int rec_size,char filchar);
VACB *open_v_array(char *filename,int buffer_size);
void close_v_array(VACB *v_array);
void *access_v_rec(VACB *v_array,long index);
```

End Listing One

Listing Two

```
Listing Two

/*  Virtual Array Access Routines  */

#include <varray.h>

#define header 7

/****************
 * init_v_array *
 ****************/

int init_v_array(filename,rec_size,filchar)
char *filename, filchar;
int  rec_size;
{
   long size;
   FILE *f;
   f = fopen(filename,"wb");
   if (f != NULL) {
      size = 0;
      fwrite(&size,4,1,f);          /* write array size of zero */
      fwrite(&rec_size,2,1,f);      /* and array element size   */
      fwrite(&filchar,1,1,f);       /* and fill char            */
      fclose(f);                    /* to file header */
      return(1);
   }
   else
      return(NULL);
}

/****************
 * open_v_array *
 ****************/

VACB *open_v_array(filename,buffer_size)
char *filename;
int  buffer_size;
{
   VACB *v_array;
   char *buf_ptr;
   int  i;
   char filchar;

   /* allocate virtual array control block */

   v_array = (VACB *) malloc(sizeof(VACB));
   if (v_array == NULL) return(NULL);

   /* open virtual array file */

   v_array->file = fopen(filename,"r+b");
   if (v_array->file == NULL) {
      free(v_array);
      return(NULL);
   };

   /* get array size and element size for control block */

   fread(&v_array->size,4,1,v_array->file);
   fread(&v_array->elsize,2,1,v_array->file);
   fread(&filchar,1,1,v_array->file);
   v_array->buf_elsize = v_array->elsize + 4;
```

(continued on page 65)

$5 SAVINGS

SUBSCRIBE AND SAVE!

Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS

and save over $5—a 15% savings off the cover price!

☐ 1 year $29.97 ☐ 2 years $56.97

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

*Savings based on the full one-year newsstand rate of $35.40. All foreign countries please add $11 per year for surface mail; Canada & Mexico add $28 per year for airmail; other countries add $32 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

A Publication of M & T Publishing, Inc.

366S

COMMENTS & SUGGESTIONS

May 1988, #139

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

______

______

______

Comments or suggestions ______

______

______

______

Name: ______

Address: ______

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
Box 3713
Escondidio, CA 92025-9843

Listing Two *(Listing continued, text begins on page 46.)*

```
      /* allocate buffer */

      v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
      if (v_array->buffer == NULL) {
         fclose(v_array->file);
         free(v_array);
         return(NULL);
      };
      v_array->buf_size = buffer_size;

      /* set up blank_rec using the fill character in array header */
      /* for initializing new array elements */

      buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
      v_array->blank_rec = buf_ptr + 4;
      for (i=0; i < v_array->buf_elsize; i++)
         *buf_ptr++ = filchar;

      /* set record index negative for all buffer elements   */

      buf_ptr = v_array->buffer;
      for (i = 0; i < v_array->buf_size; i++) {
         *((long *)buf_ptr) = -1L;
         buf_ptr += v_array->buf_elsize;
      };
      return(v_array);
   }
      /*****************
       * close_v_array *
       *****************/

      void close_v_array(v_array)
      VACB *v_array;
      {
         int  i;
         char *buf_ptr;
         long rec_index, file_offset;

         buf_ptr = v_array->buffer;

         /*  flush buffer */

         for (i=0; i < v_array->buf_size; i++) {

            /* check each element index */
            /* if element present, write it to disk */

            rec_index = *((long *)buf_ptr);
            if (rec_index >= 0) {
                 file_offset = header + rec_index * v_array->elsize;
                 fseek(v_array->file,file_offset,0);
                 fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
            };
            buf_ptr += v_array->buf_elsize;
         };
         free(v_array->buffer);       /* de-allocate buffer */
         fclose(v_array->file);       /* close array file   */
         free(v_array);               /* de-allocate VACB   */
      }

      /****************
       * access_v_rec *
       ****************/

      void *access_v_rec(v_array,index)
      VACB *v_array;
      long index;
      {
         char *buf_ptr;
         int  buf_index;
         long rec_index, temp_index;

         /* calculate buffer address of referenced element */

         buf_index = index % v_array->buf_size;
         buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
         rec_index = *(long *)buf_ptr;

         /* if element present, return its buffer address */

         if (rec_index == index) return(buf_ptr + 4);

         /* if element doesn't exist, extend the file */

         if (index >= v_array->size) {
            fseek(v_array->file,0,2);
            for (temp_index = v_array->size; temp_index++ <= index; )
                 fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
            v_array->size = index + 1;
            fseek(v_array->file,0,0);
            fwrite(&v_array->size, 4, 1, v_array->file);
         };
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 46.)*

```
     /* if buffer slot is occupied by another element, */
     /* save it to disk */

     if (rec_index >= 0) {
        fseek(v_array->file, rec_index * v_array->elsize + header, 0);
        fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
     };

     /* read referenced element into buffer slot */

     fseek(v_array->file, index * v_array->elsize + header, 0);
     fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
     *((long *)buf_ptr) = index;

     /* return address of element in buffer */

     return(buf_ptr + 4);
  }
```

```
Listing Three

/* Example Program Using Virtual Arrays */

#include <varray.h>

/* Access Macros */

#define VREC(i) ((items *)access_v_rec(item_array,i))
#define item(i) VREC(i)->v_item
#define qty(i) VREC(i)->v_qty
#define desc(i) VREC(i)->v_desc

/* Array element structure typedef */

typedef struct {
   int v_item,
       v_qty;
   char v_desc[24];
} items;

main()
{
   VACB *item_array;
   long i;

   /* create a virtual array setting element size to */
   /* the size of items structure and setting the    */
   /* initialization char to the space char          */

   init_v_array("ITEMS.VAR",sizeof(items),' ');

   /* open the virtual array, reserve buffer space for 10 elements */

   item_array = open_v_array("ITEMS.VAR",10);


   /* create 50 array items */

   for (i = 0; i < 50 ; i++) {
      item(i) = i + 1;
      qty(i)  = 0;
      sprintf(desc(i)," Item # %ld", i + 1);
   }

   /* print contents of the 50 array items */
   /* plus the ascii code of last char in v_desc */

   for (i = 0; i < 50; i++)
      printf("Element # %ld  Item = %d  Qty = %d Desc = %s  %d\n",
             i, item(i), qty(i), desc(i), (int) desc(i)[23]);

   /* close virtual array */

   close_v_array(item_array);
}
```

End Listings

Listing One *(Text begins on page 72.)*

```
#include <ctype.h>

char *factor( str )
char    *str;
{
    char *expr();

    if( isalpha( *str ) )         /* F -> name  */
        printf( "%c\n", *str );
    else if( *str == '(' )        /* F -> ( E ) */
        str = expr( ++str );
    return ++str;
}

char *term( str )
char    *str;
{
    str = factor( str );          /* T -> FT'  */
    while( *str == '*' )          /* T'-> *FT' */
    {
        str++;
        str = factor( str );
        printf( "*\n" );
    }
    return str;                   /* T'-> eps */
}

char   *expr( str )
char   *str;
{
    str = term( str );            /* E -> TE'  */
    while( *str == '+' )          /* E'-> +TE' */
    {
        str++;
        str = term( str );
        printf( "+\n" );
    }
    return str;                   /* E'-> eps */
}

main()
{
    char  buf[80];
    while( gets(buf) )
        expr( buf );
}
```

End Listing One

Listing Two

```
typedef struct node
{
    char        name[16];
    int         op;
    struct node *left;
    struct node *right;
}
NODE;

NODE *new()
{
    NODE *p;
    void *calloc();

    if( !(p = (NODE *) calloc( 1, sizeof(NODE) ) ) )
        exit(1);

    return p;
}
```

End Listing Two

Listing Three

```
NODE    *build()
{
    char   buf[80];
    NODE   *stack[ 10 ];
    NODE   **sp = stack - 1;
    NODE   *p;

    while( gets(buf) )
    {
        switch( *buf )
        {
        default:  p = new();
                  strcpy( p->name, buf );
                  *++sp = p;
                  break;

        case '*':
        case '+':
                  p         = new();
                  p->right  = *sp-- ;
                  p->left   = *sp-- ;
```

(continued on next page)

Listing Three

(Listing continued, text begins on page 72.)

```
                        p->op       = *buf  ;
                        p->name[0]  = *buf  ;
                        *++sp       = p     ;
                        break;
                }
            }
            return *sp--:
        }
```

End Listing Three

Listing Four

```
trav( root )
struct node *root;
{
    static int tnum = 0;

    if( !root )
        return;

    if( !root->op )     /* leaf */
    {
        printf ( "t%d = %s\n", tnum, root->name );
        sprintf( root->name , "t%d", tnum );
        ++tnum;
    }
    else
    {
        trav( root->left  );

        if( root->left != root->right )     /* Always true      */
            trav( root->right );            /* unless optimized */

        printf("%s %c= %s\n", root->right->name,
                            root->op, root->left->name );
        strcpy( root->name,   root->right->name );
    }
}
```

End Listing Four

Listing Five

```
optimize( root )
NODE    *root;
{
    /* Stupid optimizer to eliminate common subexpressions */

    char sig1[ 32 ];
    char sig2[ 32 ];

    if( root->right && root->left )
    {
        optimize( root->right );
        optimize( root->left  );

        *sig1 = *sig2 = '\0';
        makesig( root->right, sig1 );
        makesig( root->left , sig2 );

        if( strcmp( sig1, sig2 ) == 0 )    /* subtrees match */
            root->right = root->left ;
    }
}

makesig( root, str )
NODE    *root;

    char    *str;
    {
        if( !root )
            return;

        strcat( str, root->name );
        makesig( root->left,  str );
        makesig( root->right, str );
    }
```

End Listings

Listing One *(Text begins on page 92.)*

```
Listing One

/* MKTABLE.C: Makes a data table with header record */

#include <stdio.h>
#include <string.h>
#define  SIG  19364              /* application file signature */

typedef struct {
  char      fname [20];
  int       ftype, flen;
} DESCR;

struct {                                /* header record for file */
  unsigned  signature;
  int       nrecs;
  char      tablename [10];
  int       reclen;
  long      datastart;
  int       descrsize;
  int       ndescr;
} header;

struct {                                  /* data record for file */
  char      name [20];
  int       age;
} data;

main ()
{
FILE  *fp;
char  age [3];
int    n;
DESCR descr;

  fp = fopen ("database.xyz", "w");              /* create file */

  header.signature = SIG;                   /* initialize header */
  header.nrecs = 0;
  strcpy (header.tablename, "Age list");
  header.reclen = sizeof data;
  header.datastart = 256L;
  header.descrsize = sizeof (descr);
  header.ndescr = 2;
  fwrite (&header, sizeof header, 1, fp);    /* write to file */

  strcpy (descr.fname, "NAME");         /* initialize descriptor */
  descr.ftype = 1;
  descr.flen  = 20;
  fwrite (&descr, sizeof (descr), 1, fp);    /* write to file */

  strcpy (descr.fname, "AGE");                    /* ditto above */
  descr.ftype = 0;
  descr.flen  = 2;
  fwrite (&descr, sizeof (descr), 1, fp);

  fseek (fp, 256L, SEEK_SET);

  do {                                           /* capture data */
    printf ("\nName? ");
    gets (data.name);
    if (strlen (data.name)) {            /* continue until blank */
      printf ("Age?  ");
      gets (age);
      data.age = atoi (age);
      fwrite (&data, sizeof data, 1, fp);        /* write record */
      header.nrecs += 1;                         /* count record */
    }
  } while (strlen (data.name));          /* until no more entered */

  fseek (fp, 0L, SEEK_SET);                /* go to start of file */
  fwrite (&header, sizeof header, 1, fp);    /* update header */
  fclose (fp);                                     /* close file */
}
```

End Listing One

Listing Two

```
Listing Two

PROGRAM nonpas;

  { Reads a non-Pascal database table with a header record }
  { and some number of fixed-length data records           }

CONST signature = 19364;                  { application signature }
      divider = '--------------------------------------------';
```

(continued on next page)

Listing Two

(Listing continued, text begins on page 92.)


```
TYPE  s20              = STRING [20];
      pac              = PACKED ARRAY [1..20] OF CHAR;

  headrec = RECORD CASE tag : INTEGER OF
  1: (signature   : WORD;                        { This is the real layout }
      nrecs       : WORD;                              { # data records }
      placeholdr  : PACKED ARRAY [1..10] OF CHAR;      { table name }
      reclen      : INTEGER;                       { data record length }
      datastart   : LONGINT;                     { file offset for data }
      descrsize   : INTEGER;                   { field descriptor size }
      ndescr      : INTEGER);               { number of fields per rec }
  2: (dummy1,
      dummy2      : WORD;
      tablename   : pac);                         { To fool typechecking }
  3: (stream      : PACKED ARRAY [1..24] OF BYTE);
  END;

  fieldrec = RECORD CASE tag : INTEGER OF
  1: (fname       : pac;
      ftype       : INTEGER;
      flen        : INTEGER);
  2: (stream      : PACKED ARRAY [1..24] OF BYTE);
  END;

VAR   header    : headrec;
      field     : ARRAY [1..10] OF fieldrec;         { descriptors }
      n         : INTEGER;
      table     : FILE OF BYTE;
{ -------------------------- }

FUNCTION asciiz (max : INTEGER; VAR strng : pac) : s20;

    { Returns a Pascal string from a null-terminated string
      that is <= max bytes long }

VAR   i      : INTEGER;
      result : STRING [20];

BEGIN
  result := '';
  FOR i := 1 TO max DO
    IF strng [i] <> CHR (0) THEN
    result := result + strng [i];
  asciiz := result;
END;
{ -------------------------- }

PROCEDURE getDescriptors;

    { Reads field descriptors from header record }

VAR   c, d : INTEGER;

BEGIN
  FOR d := 1 to header.ndescr DO
    FOR c := 1 TO header.descrsize DO
      READ (table, field [d].stream [c]);
END;
{ -------------------------- }

PROCEDURE showHeaderInfo;

    { List information about the file format }

VAR   d : INTEGER;

BEGIN
  WRITELN (divider);
  WRITELN ('Table name is ',
           asciiz (10, header.tablename));
  WRITELN ('Table contains ', header.nrecs, ' records');
  WRITELN ('Data record length in bytes is ',
           header.reclen);
  WRITELN ('Each record contains ', header.ndescr, ' fields:');
  getDescriptors;
  FOR d := 1 TO header.ndescr DO BEGIN
    WRITELN ('   Field name:      ', asciiz (20, field [d].fname));
    WRITE   ('   Data type:       ');
    CASE field [d].ftype OF
      0: WRITELN ('Integer');
      1: WRITELN ('Character');

        END;
        WRITELN ('   Length:         ', field [d].flen);
        WRITELN;
```

```
    END;
    WRITELN ('Data records follow:');
    WRITELN;
  END;
  { -------------------------- }

  PROCEDURE showData;

        { List contents of each data record by fieldname }

  TYPE  int = RECORD CASE tag : INTEGER OF
          1: (number : INTEGER);
          2: (stream : PACKED ARRAY [1..2] OF BYTE);
        END;

  TYPE  charfield = RECORD CASE tag : INTEGER OF
          1: (bf : PACKED ARRAY [1..20] OF BYTE);
          2: (cf : pac);
        END;

  VAR   rec, descr, n : INTEGER;
        intfield      : int;                      { integer data field }
        chfield       : charfield;              { character data field }

  BEGIN
    FOR rec := 1 TO header.nrecs DO                     { For each record }
      FOR descr := 1 TO header.ndescr DO BEGIN        { For each field }
        WRITE (asciiz (20, field [descr].fname));       { Show name }
        FOR n := LENGTH (asciiz (20, field [descr].fname)) TO 25 DO
          WRITE (' ');                               { cosmetic spacing }
        CASE field [descr].ftype OF
          0: BEGIN
           FOR n := 1 TO 2 DO
             READ (table, intfield.stream [n]);       { get int field }
           WRITELN (intfield.number);
          END;
           1: BEGIN
           FOR n := 1 TO field [descr].flen DO
             READ (table, chfield.bf [n]);     { get character field }
           WRITELN (asciiz (20, chfield.cf));
          END;
          END;
        END;
  END;
  { -------------------------- }

  BEGIN
    ASSIGN (table, 'DATABASE.XYZ');                       { open table }
    RESET (table);
    FOR n := 1 TO 24 DO                             { read header record }
      READ (table, header.stream [n]);
    IF signature <> header.signature THEN
      WRITELN ('File not in proper format. Program ended.')
    ELSE
      BEGIN
        showHeaderInfo;                        { Show info about the file }
        SEEK (table, header.datastart);          { go to start of data }
        showData;                              { List each record's data }
      END;
    CLOSE (table);
  END.
```

End Listings

Postfix Notation and Common-Subexpression Elimination

Most compilers don't generate binary executable code; rather, they create a program in an "intermediate language" that is translated into binary by a "compiler back end." This approach has three advantages. First, the same front end can generate code for many target machines by providing several back ends. Second, and conversely, compilers for several different languages can generate the same intermediate code, which can then be used by a single back end. And finally, intermediate code is usually easier to optimize than assembly language is.

It's the third advantage that's the main topic of this month's column: a small compiler along with a simple optimizer that does common-subexpression elimination. The code this month is not all that useful as it stands. For one thing I've left out most of the error checking to make it more understandable. Nonetheless, the concepts are quite useful and apply to several applications apart from compiler design.

One common intermediate language that compilers use is a postfix or reverse-Polish notation. Users of Hewlett-Packard calculators and Unix's dc desk-calculator program will already be familiar with the process. In postfix notation, operands are pushed onto a stack without modification. Operators, however, modify the top few items on the stack. The C fragment A*B+A*B, for example, generates the following postfix operations:

by Allen Holub

push A
push B
pop two items, multiply them, push the result
push A
push B
pop two items, multiply them, push the result
pop two items, add them, push the result

When the input is processed, the result will be on the top of the stack.

A postfix intermediate language is easy to generate because the compiler doesn't have to worry about assigning temporary variables for the rvalues; it just uses the stack as its scratch space. The second advantage is that the optimization pass can reconstruct the entire parse tree—or to be more exact, a compacted form of the parse tree called a syntax tree—from the list of instructions. It turns out that several optimizations are much easier to perform on a syntax tree than on a quad representation of the same program.

Let's look at a concrete example. The earlier expression *(A*B)+(A*B)* can be represented in the following postfix form:

A
B
*
A
B
*
+

There's no need for an explicit *push* operator as long as the operators can be distinguished from variable names. Similarly, explicit parentheses are never necessary because the order of evaluation is determined by the sequence of operations.

The set of subroutines in Listing 1, page 67, form a small recursive-decent compiler that takes as input expressions involving single-character variable names, plus signs, times signs (asterisks), and parentheses. The program outputs the input expression in postfix notation. Multiplication is of higher precedence than addition, and expressions associate left to right. Given the expression *(A*B)+(A*B)*, it produces the output described earlier. The grammar is shown in Table 1, below.

The next step is to recreate the syntax tree from the postfix expression output by the parser (see Figure 1, page 73). Note that all internal nodes represent operators and all the leaves are variable references. Also, the grouping of operators and operands is as you would expect, given the original parenthesized expressions, even though there were no parentheses in the postfix expression itself.

Before demonstrating how to do this reconstruction, I need a data structure to represent the nodes in the tree. Listing 2, page 67, shows this data structure—*NODE*—and a constructor subroutine that makes new nodes—*new( )*. The *NODE* structure is a normal binary-tree node,

expr	<– term expr'	Implemented in expr( )
expr	<– + term expr'	"
	<– ε	"
term	<– factor term'	Implemented in term( )
term'	<– * factor term'	"
	<– ε	"
factor	<– **name**	Implemented in factor( )
	<– (expr)	"

Table 1: *Grammar used by a postfix compiler*

having left and right children. In addition, the *name* field holds variable names (*A* and *B* in this case) or the operator if the node is an internal node. (The contents of this field will be modified by the optimizer, however.) The *op* field holds the operator (* or +) or is set to 0 on leaf nodes.

The *build()* subroutine, shown in Listing 3, page 67, creates a syntax tree from a postfix input file generated by the parser I just discussed. The input file must have one operand or operator per line, and it must be perfect—that is, in order to simplify the code, I've dispensed with error detection. Input lines are read from standard input, and the subroutine returns a pointer to the root node of the tree.

Trees are built in a bottom-up fashion, using a local stack to keep track of the partially constructed tree. The *default* case is executed for variable names. It allocates and initializes a new node and then pushes a pointer to the new node onto the stack. The child pointers are initialized to *NULL* by *new()*.

Operators are handled differently because they're internal nodes. A new node is allocated and initialized, then pointers to two existing nodes are popped and the child pointers of the new internal node are made to point at these. Finally, the new node is pushed.

The tree for the input discussed earlier is built as shown in Figure 2, this page. Code can be generated from this tree by doing a depth-first traversal (visit the children, then the parent). At every lvalue (that is, variable reference), generate a temporary=variable instruction. At every internal node, generate the code necessary to perform the operation on the temporaries that resulted from traversing the previous level, putting the result into a new temporary.

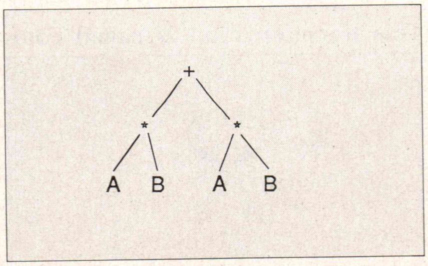


Figure 1: *The syntax tree*

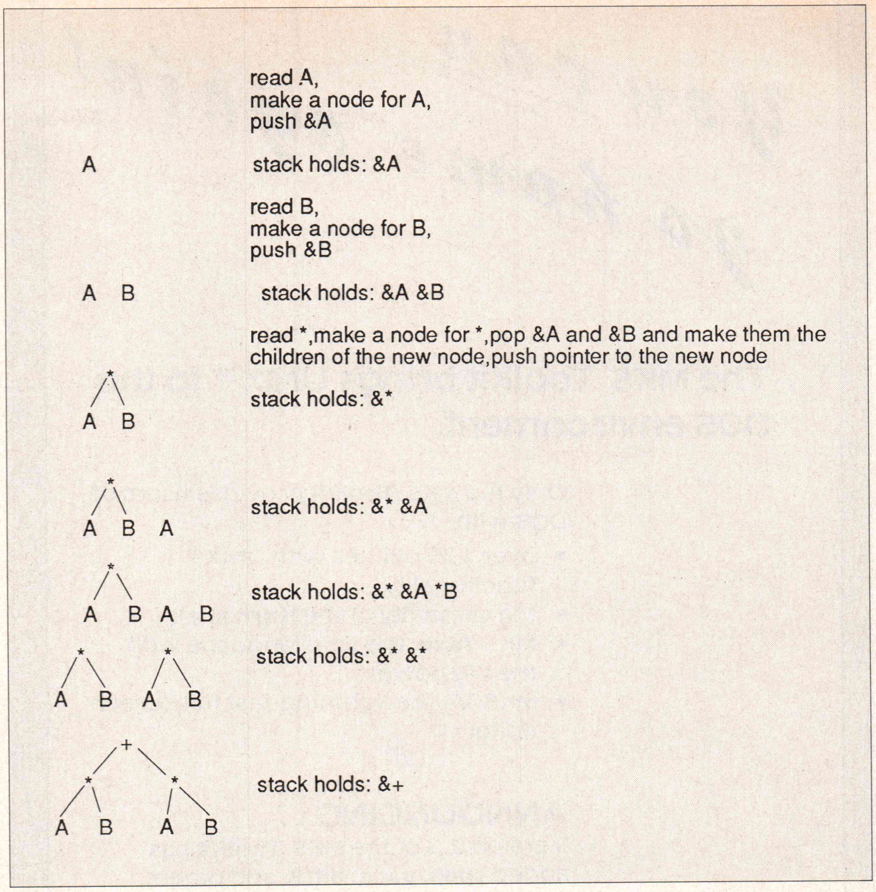


Figure 2: *The input tree*

C CHEST
(continued from page 73)

That is, the previously constructed tree will generate the following output:

```
t0 = A
t1 = B
t1 *= t0
t2 = A
t3 = B
t3 *= t2
t3 += t1
```

The *trav( )* subroutine in Listing 4, page 67, does the traversal. It takes the pointer returned from the previous *build( )* call as its initial argument. If *root->op* is zero, then the current node is a leaf, and you generate the code to move it to a temporary variable. The *sprintf( )* call overwrites the *name* field with the name of the temporary variable. If the *op* field is nonnull, you're processing an interior node. In this case, you do an in-order traversal. The *if* statement is always true (for now—things will change momentarily). The following *printf( )* call prints the instruction, using the *name* fields of the two children to find out what temporaries to use. The *strcpy( )* call then overwrites the *name* field of the current node to reflect the temporary that got the result of the last operation.

The code that *trav( )* outputs isn't too great, primarily because the subexpression *A*B* is evaluated twice. It would be better to perform the multiply only once and use the temporary rvalue generated by that multiply twice. That is, you'd like to generate the following output:

```
t0 = A
t1 = B
t1 *= t0
t1 += t1
```

This transformation is called *com-*

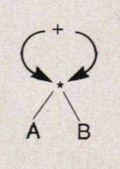


Figure 3: *The new syntax tree*

UNLEASH YOUR 80386!

Your 80386-based PC should run two to three times as fast as your old AT. This speed-up is primarily due to the doubling of the clock speed from 8 to 16 MHz. The new MicroWay products discussed below take advantage of the real power of your 80386, which is actually 4 to 16 times that of the old AT! These new products take advantage of the 32 bit registers and data bus of the 80386 and the Weitek 1167 numeric coprocessor chip set. They include a family of MicroWay 80386 compilers that run in protected mode and numeric coprocessor cards that utilize the Weitek technology.

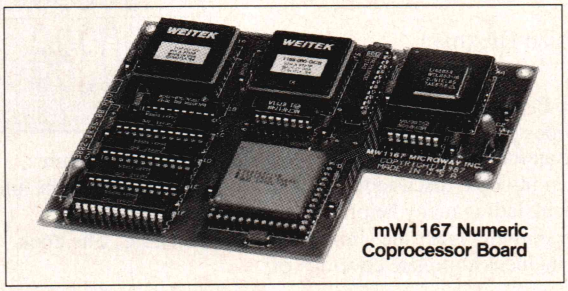
The benefits of our new technologies include:

- An increase in addressable memory from 640K to 4 gigabytes using MS-DOS or Unix.
- A 12 fold increase in the speed of 32 bit integer arithmetic.
- A 4 to 16 fold increase in floating point speed over the 80387/80287 numeric coprocessors.

Equally important, whichever MicroWay product you choose, you can be assured of the same excellent pre- and post-sales support that has made MicroWay the world leader in PC numerics and high performance PC upgrades. For more information, please call the Technical Support Department at

617-746-7341

After July 1988 call 508-746-7341



mW1167 Numeric Coprocessor Board

MicroWay® 80386 Support

MicroWay 80386 Compilers

NDP Fortran-386 and **NDP C-386** are globally optimizing 80386 native code compilers that support a number of Numeric Data Processors, including the 80287, 80387 and mW1167. They generate mainframe quality optimized code and are syntactically and operationally compatible to the Berkeley 4.2 Unix f77 and PCC compilers. MS-DOS specific extensions have been added where necessary to make it easy to port programs written with Microsoft C or Fortran and R/M Fortran.

The compilers are presently available in two formats: Microport Unix 5.3 or MS-DOS as extended by the Phar Lap Tools. MicroWay will port them to other 80386 operating systems such as OS/2 as the need arises and as 80386 versions become available.

The key to addressing more than 640 kbytes is the use of 32-bit integers to address arrays. NDP Fortran-386 generates 32-bit code which executes 3 to 8 times faster than the current generation of 16-bit compilers. There are three elements each of which contributes a factor of 2 to this speed increase: very efficient use of 80386 registers to store 32-bit entities, the use of inline 32-bit arithmetic instead of library calls, and a doubling in the effective utilization of the system data bus.

An example of the benefit of excellent code is a 32-bit matrix multiply. In this benchmark an NDP Fortran-386 program is run against the same program compiled with a 16-bit Fortran. Both programs were run on the same 80386 system. However, the 32-bit code ran 7.5 times faster than the 16-bit code, and 58.5 times faster than the 16-bit code executing on an IBM PC.

NDP FORTRAN-386™ $595
NDP C-386™ $595

MicroWay Numerics

The **mW1167™** is a MicroWay designed high speed numeric coprocessor that works with the 80386. It plugs into a 121 pin "Weitek" socket that is actually a super set of the 80387. This socket is available on a number of motherboards and accelerators including the AT&T 6386, Tandy 4000, Compaq 386/20, Hewlett Packard RS/20 and MicroWay Number Smasher 386. It combines the 64-bit Weitek 1163/64 floating point multiplier/adder with a Weitek/Intel designed "glue chip". The mW1167™ runs at 3.6 MegaWhetstones (compiled with NDP Fortran-386) which is a factor of 16 faster than an AT and 2 to 4 times faster than an 80387.

mW1167 16 MHz $1495
mW1167 20 MHz $1995

Monoputer™ – The INMOS T800-20 Transputer is a 32-bit computer on a chip that features a built-in floating point coprocessor. The T800 can be used to build arbitrarily large parallel processing machines. The Monoputer comes with either the 20 MHz T800 or the T414 (a T800 without the NDP) and includes 2 megabytes of processor memory. Transputer language support from MicroWay includes Occam, C, Fortran, Pascal and Prolog.

Monoputer T414-20 with 2 meg[1] ... $1495
Monoputer T800-20 with 2 meg[1] ... $1995

Quadputer™ can be purchased with 2, 3 or 4 transputers each of which has 1 or 4 megabytes of memory. Quadputers can be cabled together to build arbitrarily fast parallel processing systems that are as fast or faster than today's mainframes. A single T800 is as fast as an 80386/mW1167 combination!

Biputer™ T800/T414 with 2 meg[1] $3495
Quadputer 4 T414-20 with 4 meg[1] ... $6000

[1] Includes Occam

80386 Multi-User Solutions

AT8™ – This intelligent serial controller series is designed to handle 4 to 16 users in a Xenix or Unix environment with as little as 3% degradation in speed. It has been tested and approved by Compaq, Intel, NCR, Zenith, and the Department of Defense for use in high performance 80286 and 80386 Xenix or Unix based multi-user systems.

AT4 – 4 users $795
AT8 – 8 users $995
AT16 – 16 users $1295

Phar Lap™ created the first tools that make it possible to develop 80386 applications which run under MS-DOS yet take advantage of the full power of the 80386. These include an 80386 monitor/loader that runs the 80386 in protected linear address mode, an assembler, linker and debugger. These tools are required for the MS-DOS version of the MicroWay NDP Compilers.

Phar Lap Tools $495

PC/AT ACCELERATORS

287Turbo-10 10 MHz $450
287Turbo-12 12 MHz $550
287TurboPlus-12 12 MHz $629
FASTCACHE-286 9 MHz $299
FASTCACHE-286 12 MHz $399
SUPERCACHE-286 .... $499

MATH COPROCESSORS

80387-20 20 MHz $895
80387-16 16 MHz $495
80287-10 10 MHz $349
80287-8 8 MHz $259
80287-6 6 MHz $179
8087-2 8 MHz $154
8087 5 MHz $99

Micro Way

The World Leader in PC Numerics

P.O. Box 79, Kingston, Mass. 02364 USA (617) 746-7341
32 High St., Kingston-Upon-Thames, U.K., 01-541-5466
St. Leonards, NSW, Australia 02-439-8400

CIRCLE NO. 162 ON READER SERVICE CARD

More Rvalues and Implicit Type Conversion

This month's Flotsam and Jetsam continues the subject of rvalues started last month by looking at another problem caused by rvalues and the automatic type-conversion rules. The following code does not work properly:

```
long x;
int y = 32767, z = 32767;
x = y * z;
```

If your machine has a 16-bit *int*, *x* will have the value 1 assigned to it.

The problem here is rvalues—temporary variables that are used by the compiler to hold the intermediate steps of an expression evaluation. The compiler interprets the previous code using the following steps:

1. Copies *y* to the *int*-size temporary *t0*
2. Copies z to the *int*-size temporary *t1*
3. Multiplies *t0* by *t1* and puts the result into the *int*-size temporary *t2*
4. Converts *t2* to *long* and puts the result into the *long*-size temporary *t3*
5. Copies *t3* to *x*

Because the result of the multiplication is stored in an *int*-size temporary, the product will be truncated. That is, 32,767 * 32,767 should yield 0x3fff0001, the top four digits of which will be truncated as part of the assignment.

The problem is that the compiler has no general understanding of the expression as a whole; rather, it knows only about a single operator and the associated operands. It breaks up the foregoing expression into *(x = (y * z))* and then starts evaluating from the inside out—that is, it starts with the multiplication. Because *y* and z are both of type *int*, the result of evaluating the subexpression is put into an *int*-size rvalue. Only after the multiplication is performed will the compiler move out a nesting level and apply the = operator. Now the operands will be *x* and the rvalue that resulted from the previous subexpression. Noticing that *x* is of type *long* and that the temporary is of type *int*, the compiler will correctly convert the *int* to *long* before doing the assignment.

All this is not a bug—the compiler is just doing what you told it to do. You can fix the problem in several ways. The easiest is to declare either *x* or *y* (or both) as *long*. This way the conversion to *long* is moved up to an earlier stage of the evaluation process. Assuming that you've made *y* a *long*, the compiler will do the following:

1. Copy *y* to the *long*-size temporary *t0*
2. Copy z to the *int*-size temporary *t1*
3. Noticing that the two operands are of different types, convert the *int*-size *t1* to the *long*-size temporary *t2*
4. Multiply *t0* by *t2* and put the result into the *long*-size temporary *t3*
5. Copy *t3* to *x*

Alternatively, you could use the cast operator, which creates a temporary variable that has the specified type and then copies the operand to the new variable. You can use any of the following:

```
x = (long)y * z;
x = y * (long)z;
x = (long)y * (long)z;
```

Remember, here, that the cast is an operator. That is, it's executed at run time (unless the optimizer can fix things), and it creates an rvalue of a given type and copies something into that rvalue, doing any necessary type conversions along the way.

You form a cast by writing out a declaration of a variable having the desired type, surrounding the declaration with parentheses, and removing the name and semicolon. For example, to create a cast for a pointer to an array of structures of type *building*, you can use the following procedure:

```
struct building (*p)[10];
(struct building (*p)[10];)
(struct building (*)[10])
```

Note that the parentheses around **p* are required here because *p* is a pointer to an array of structures (as compared to an array of pointers to structures). The parentheses are needed because the brackets are of higher precedence than the start, so the default binding (without parentheses) is *struct building *(p[10]);*.

—*A.H.*

C CHEST
(continued from page 74)

mon-subexpression elimination. The optimization is performed by analyzing and then modifying the syntax tree. Because both subtrees of the + node are identical, the optimizer can eliminate one subtree and make both pointers in the + node point at the remaining subtree. The new syntax tree looks like that in Figure 3, page 74. That is, both pointers in the + node point at the * node. This modified data structure is called a *directed acyclic graph*, or *DAG*.

The DAG is created from the syntax tree by the *optimize()* function in Listing 5, page 68. This routine traverses the interior nodes of the tree, comparing the two subtrees. If the subtrees are identical, the left and right pointers of the parent node are made to point at the same child, effectively removing the other child from the tree. The comparison is done using the *makesig()* function, which traverses an entire subtree, assembling a string that shows the in-order traversal of the subtree by concatenating all the *name* fields. For example, the original syntax tree, when traversed from the root, creates the signature string +***ABAB*. If two subtrees generate the same signature, they're equivalent.

If two subtrees generate the same signature, they're equivalent.

Finally, you can traverse the DAG using the same *trav()* function that you used earlier. That *if* statement will now come into play, however, preventing you from traversing the common subtree twice.

Bibliography

This month's column is an excerpt

C CODE FOR THE PC

source code, of course

Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
CQL Query System (SQL retrievals plus windows)	$325
GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$295
Aspen Software PC Curses (System V compatible, extensive documentation)	$250
Vitamin C (MacWindows)	$200
TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
Essential resident C (TSRify C programs, DOS shared libraries)	$165
Essential C Utility Library (400 useful C functions)	$160
Essential Communications Library (C functions for RS-232-based communication systems)	$160
Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
Greenleaf Functions (296 useful C functions, all DOS services)	$150
OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
American Software Resident-C (TSRify C programs)	$130
PC Curses Package (full System V, menu and data entry examples)	$120
CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
Minix Operating System (U**x-like operating system, includes manual)	$105
PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
The Profiler (program execution profile tool)	$100
Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$95
C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
Professional C Windows (windows and keyboard functions)	$80
JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
WKS Library (C program interface to Lotus 1-2-3 program & files)	$80
ME (programmer's editor with C-like macro language by Magma Software)	$75
Professional C Windows (lean & mean window and keyboard handler)	$70
Quincy (interactive C interpreter)	$60
EZ_ASM (assembly language macros bridging C and MASM)	$60
PTree (parse tree management)	$60
HELP! (pop-up help system builder)	$50
Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
Heap Expander (dynamic memory manager for expanded memory)	$50
Make (macros, all languages, built-in rules)	$50
Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
Coder's Prolog (inference engine for use with C programs)	$45
C-Help (pop-up help for C programmers ... add your own notes)	$40
Biggerstaff's System Tools (multi-tasking window manager kit)	$40
CLIPS (rule-based expert system generator, Version 4.0)	$35
TELE Kernel or TELE Windows (Ken Berry's multi-tasking kernel & window package)	$30
Clisp (Lisp interpreter with extensive internals documentation)	$30
Translate Rules to C (YACC-like function generator for rule-based systems)	$30
6-Pack of Editors (six public domain editors for use, study & hacking)	$30
ICON (string and list processing language, Version 6 and update)	$25
LEX (lexical analyzer generator)	$25
Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
AutoTrace (program tracer and memory trasher catcher)	$25
C Compiler Torture Test (checks a C compiler against K & R)	$20
Benchmark Package (C compiler, PC hardware, and Unix system)	$20
TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
A68 (68000 cross-assembler)	$20
List-Pac (C functions for lists, stacks, and queues)	$20
XLT Macro Processor (general purpose text translator)	$20

Data

WordCruncher (text retrieval & document analysis program)	$275
DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
Webster's Second Dictionary (234,932 words)	$60
U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12

Free shipping on prepaid orders

For delivery in Texas add 7%

MasterCard/VISA

C CHEST
(continued from page 78)

from a forthcoming book (by myself) on compiler design, to be published by Prentice-Hall in late 1988 (or early 1989).

You can also refer to the following books:

Aho, Sethi, and Ullman. *Compilers: Principles, Techniques, and Tools*. Reading, Mass.: Addison-Wesley, 1986. Pages 25-82 have a discussion about recursive-decent parsing, and pages 279-387 contain more information about advanced optimization techniques using DAGs.

Gries, David. *Compiler Construction for Digital Computers*. New York: Wiley, 1971: 376-411. This book also has a discussion about recursive-descent parsing.

Holub, Allen. *The C Companion*. Englewood Cliffs, N.J.: Prentice-Hall, 1987: 189-212. This book contains an introductory-level discussion of recursion in general and recursive-decent parsers in particular.

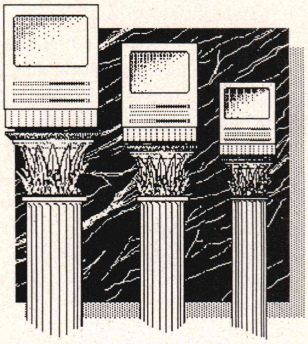
Sarff, Gary. "Optimization Strategies." *Computer Language*, vol. 2, no. 12 (December 1985): 27-32. This article has a good discussion of simple peephole-optimization techniques.

DDJ

(Listings begin on page 67.)

Vote for your favorite feature/article.
Circle Reader Service **No. 8.**

A Macworld Expo Potpourri and the Scouting Toolkit Wrap-Up



I almost didn't go to January's Macworld Expo in San Francisco. I was busy, I was tired, I could read about it later, there was so much to keep learning right here at home, various project deadlines were screaming, late 20th century cities are filled with nuts, it was already too late for one of the really good parties, and on and on—the typical list of yattas. But something inside whispered, "Go." I pay attention to that voice; otherwise it slaps me silly. I went.

Expo in a Nutshell

Quick bottom line: The Mac has won, and this Expo was the happy celebration.

Reasons for victory? Hmm.... How about snap and sizzle? It just feels good to use a Mac. There's some great Mac software. The hardware's gained a lot of horsepower. Thanks go to a lot of people: Xerox PARC, the original Apple Mac team, the early buyers, Apple's tech writers and support teams, all the programmers who've survived the steep learning curve, David E. Smith and his stable of MacTutor writers, Motorola's 680x0 chip design teams, all those who opened the hardware, Jobs, Sculley, and on and on beyond the spatial limits of this column.

The crowd energy at the Expo was a big wave, with as fine a glassy-walled tube to hang a cybernetic ten within as any mass computer event I've attended. There was too much to see, too many people to talk to,

by Stan Krute

and some fun parties with good talk and/or loud, danceable music. All in all, a mighty fine fourth birthday.

Hackus Triumphus

The Mac made one of its first big public appearances at the 1984 West Coast Computer Faire. While talking to an Apple representative at that event, I yammered about memory, hard disks, slotlessness, and printer limitations. He scanned me up and down. Our disguises are easily pierced. "You're a hacker. Those are just hacker issues. Normal users don't need that stuff. We're giving them what they need."

So the Mac took off fast, then almost died. More RAM, SCSI drives, the Mac II, and the LaserWriter, in synergy with some innovative third-party software, brought it back to life and success. One of the treats of the Expo was seeing so many of the people who contributed to that triumph, the once-maligned hackers, now sitting in decision-making positions at Apple and the third-party development companies. Apple's in a particular hacker-hiring frenzy these days, picking up some great Mac people at a torrid pace. Sweet success over time is the best retort to silliness, eh?

Your Code, Your Skills

Looking for places other than Apple to sell all those great Mac programs and programming skills you're busy developing? As I roamed the Expo, I continually asked company representatives if they were looking to buy. I'm not your humble servant for nothing. They were. The consensus: Competent Mac programmers are still a rare commodity. Especially ones who are able/willing to write robust programs that provide specific solutions in very Mac-like ways and don't mind reworking their code in response to intelligent critiques.

So get copies of the Expo guide and/or the leading magazines so you'll have some addresses, crank up the word processor and mail-merge software, and send out those inquiry letters—the window won't be open this wide forever.

Buy This Book

Scott Knaster was at the Expo. He's the Mac tech-support whiz who left Apple for a start-up stint with Guy Kawasaki at Acius. He's now back at Apple, deep into future stuff, and has a new book out: *Macintosh Programming Secrets*, published by Addison-Wesley.

Scott's been immersed in Mac programming for a long time and knows a *lot* about the art and science of it. There may be someone who's examined and debugged more Mac source code from more different programmers than Scott has, but I doubt it. His book pulls together a wide range of important, state-of-the-art, Macintosh programming concepts. It's got grand overviews, detailed techniques, and useful summaries. A lot of stuff you just won't see anywhere else. Plus, Scott explains things well. His writing style's smooth. He throws in history, cartoons, funny jokes, and even a special cameo appearance by Our Nation's Leader.

Sometimes, usually around 3:00 in the morning, I think I know what's going on inside the Mac. The thought has a short half-life. This book gives a more permanent effect and works in the bright light of day. It's one of those products that'll let you enjoy getting smarter. Back in the January column, I gave a short list of books that form the core of a good Mac programming library. Add this one to the group, near the top.

News from APDA

I wanted to talk with Dave Lingwood and Dick Hubert, a couple of

BREAKTHROUGH

COMMON LISP FOR MICROCOMPUTERS

Develop and deliver professional AI applications on the Macintosh with Allegro CL for only $600.

We're revolutionizing the economics of AI.
With Allegro Common LISP, you can develop your applications in a first-class environment and deliver them on the lowest-priced platform ever! For only $600, Allegro CL brings the complete Common Lisp standard and the best features of high-end LISP environments to the entire Apple Macintosh family of personal computers.

You Get a Full Common Lisp
Unlike other implementations of "Common LISP" for the Macintosh and IBM PC, Allegro CL is truly complete. No features have been left out. Allegro CL delivers lexical closures, complex numbers, adjustable arrays, and all other features of Common Lisp as specified in *Common LISP: the Language* by Guy Steele Jr.

With the Best Environment.
The Allegro CL programming environment is completely integrated with the Macintosh user interface to give you tools and features previously found only on high-end LISP machines. A programmable EMACS-style editor and other tools—such as a stepper, debugger, and a window based inspector—help you get your programs up and running quickly and easily.

Ask any of our customers, many of them major corporations and leading-edge companies in AI research. They'll tell you that Allegro CL for the Macintosh is the best programming environment this side of $60,000.

Allegro CL is Fast, Compact
Until now, the smallest Common LISP for a microcomputer was 3 megabytes of code. On workstations, Common LISPs routinely occupy between 5 and 8 megabytes. Allegro CL changes all that: it fits on a single 800K floppy and runs on Macintoshes with just 1 megabyte of RAM!

Our tightly written code coupled with our incremental compiler make Allegro blaze. Many developers report compile times on a Mac II that bettered LISP machines and left other micros far behind.

And Features Advance Interface Tools.
Within a matter of days, you can have a full working model of your program's user interface. Allegro CL provides a complete set of Macintosh interface components as pre-defined, self-managing objects. You also get full low-level access to the Macintosh Toolbox. Easier implementation means you can spend more time on design and refinement. Your product—and your customers—benefit.

You Also Get Our Commitment to Continued Development.
At Franz Inc., we're in the business of bringing you leading edge technologies on accessible hardware. To that end, our first additions to Allegro CL are already available.

The Allegro Flavors module gives you compatibility with Flavors release 6.1 code libraries. The Allegro Foreign Function Interface lets you call routines written in C, Pascal, or Assembly language.

It's Yours For Next to Nothing.
We are proud to offer Allegro Common LISP at an exceptional price. Call us today. Tomorrow, you could be testing the full capabilities of our revolutionary Common LISP environment.

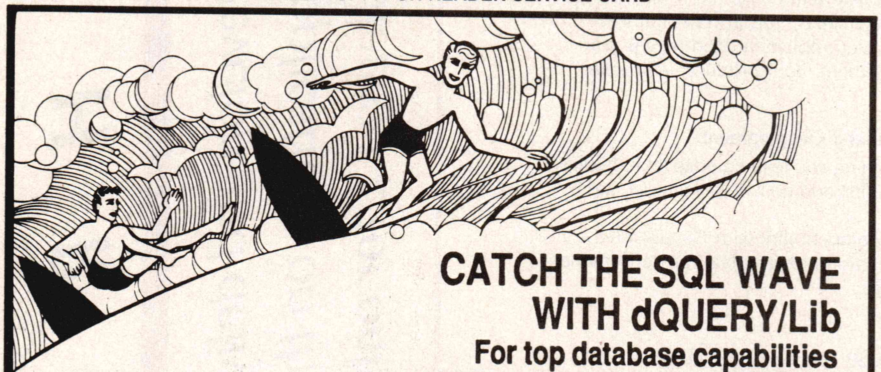
For more information on how you can get started in Artificial Intelligence on the hottest micro in the market, contact us at:

Franz Inc., 1995 University Ave., Berkeley, CA 94704
(415) 548-3600. For orders and info,
call (800) 33 FRANZ (333-7260)

Coral Software Corp., P.O. Box 307, Cambridge, MA 02142,
(617) 547-2662

FRANZ INC.
Intelligent Software Tools

CIRCLE NO. 130 ON READER SERVICE CARD



TO THE MACS
(continued from page 82)

longtime Apple communitarians and leaders, respectively, of the Apple Programmer's and Developer's Association and its mother ship, the A.P.P.L.E. Co-op. But they were always in the midst of a continuous feeding frenzy at the APDA booth.

Luckily, I bumped into Frank Catalano, the knowledgeable APDA public relations manager. APDA's up to around 20,000 members. It's got its typical order turnaround time down from 5–7 days to 48 hours. *APDAlog*, the quarterly newsletter/catalog, keeps improving. I've got the January 1988 issue in front of me, and it's fun: good articles that are worth reading and lots of new products to slaver over, including final MPW 2.0.2 tools, a LaserWriter IISC reference, and 14 pages of third-party software tools and books. MPW's pretty remarkable stuff. (Yep, I've bitten the bullet and started to dive in; reports forthcoming, as I always seem to say.)

Of special interest to HyperTalkers: APDA's *HyperCard Technical Reference*, mentioned in the March column, has been enlarged, modified, and split into two pieces. New piece number 1, *HyperCard Script Language Guide*, is Apple's definitive 200-page Hypertalk reference manual. This is a final APDA draft, about 50 percent larger than the earlier (August 11, 1987) draft. New piece number 2, *HyperCard Developer's Toolkit*, contains several goodies. There's a nice typeset document, *HyperCard Stack Design Guidelines*, from Apple's wide-awake Human Interface Group. It lays out the basic principles of intelligent stack design in a coherent way. There's a long MacWrite document that provides the details of writing XCMDs and XFCNs, along with several very useful examples in C and Pascal. There are five sample stacks that illustrate using the included XCMDs to cope with serial-port communications, sounds, and videodisc players. Finally, they've tossed in three sounds: a clang, a flute, and a weird voice saying, "Hi, there." If you're serious about HyperCard development, you'll find both these packages vital.

Finally, if you've got a book or

software tool that'd be of interest to other Mac programmers, send it to APDA for possible inclusion in *APDAlog*. APDA scans carefully, and the good stuff gets in. Not that any of us think of money (heh heh), but you could do a lot worse than to land something in this highly targeted publication.

Tools Keep Evolving, As Do Their Users

Yaaarrrggghh, always running out of time and space. Good ol' Tyler'll have my head. But, before I get down to some code talk: I was able to speak to representatives from Apple, Borland, Think (now of Symantec), Coral, and Smethers Barnes about all their hot new programming tools. Latest versions have hit the streets, as Apple's finished the current batch of major header-file changes. Source-code-level debuggers are coming. Levels of performance and abstraction are rising. The competition's hot. I was impressed by the commitment that these companies are bringing to the programming tools market.

And the programmers! *Ai carambacita!* I wish a few Martian anthropologists could've followed the crew of Macahologists who snaked through the San Francisco streets toward a Szechuan Chinese foodery Saturday night for what's become a main Macworld Expo happening—the Netters' Dinner. Combustible comestibility for a group of cybernauts who usually meet at long electronic distances. Hard to tell which was hotter/faster: the food, its consumption, or the idea flow. What is the link between this kind of cuisine and the programming mind, anyways?

Code Corner

Alright, chitchat's over. Let me calm down by discussing some highlights of the HyperCard Scouting Toolkit (ST) project whose images and code sources were shown in April. Please refer to that column for figures, listings, and an introduction to the project.

A Quick Review

Stacks are composed of one or more cards. Backgrounds hold features that are common to one or more cards. Cards can contain buttons and text fields and graphic designs. Backgrounds can also contain buttons and text fields and graphic designs. Several stacks may be linked by buttons so that they form a set of stacks.

HyperTalk programs consist of a series of message handlers. Things happen when you work with HyperCard, and those events cause messages to be sent to various HyperCard objects: buttons, fields, cards, backgrounds, stacks, external code resources, and the HyperCard program itself. A message passes up a hierarchy leading from simple objects to increasingly powerful ones, until one of those objects deals with the message by handling it and not passing it on. An object's message handlers are contained in its script. Take a look at the various HyperCard books, documentation sets, and help stacks for diagrams of that hierarchy and more detailed descriptions of the flow of control.

If you use the Scouting Toolkit for a while, you'll want to tweak it.

The Stack, the Background, and the Card

The Scouting Toolkit lives on a stack that has one background and one card. The stack has no script. Neither does the background. And the background contains neither fields nor buttons nor graphics. Backgrounds are useful in stacks with more than one card, where they are used to hold information common to several cards. In this simple stack, there's no need for the background to do anything.

The ST card, which I'll call the STBC (Bat Cave), contains 17 buttons and 2 fields. The STBC has a script with two message handlers. Refer to last column's Figure 5 for a picture of the card and last column's Listing One for the script (as well as the scripts of all the other objects in the stack and brief descriptions).

When the card is first opened up, it's sent an *openCard* message. I'm nutty about Zen-like workspaces, so this card's *openCard* handler hides several things: the menu bar, the message box, the tool window, and the pattern window. Later on, buttons on the toolkit let you toggle the visibility of those items. Minor problem in toggling: There are HyperCard functions that tell whether the last three are visible, but no such function exists for the menu bar. So the *openCard* handler stores an indicator in a global variable—one that's known throughout the stack world—imaginatively named *menubarState*.

If someone plays around with the objects on the STBC, certain buttons might go into hiding. A click anywhere in the STBC outside a button will be caught by the card's *mouseUp* message handler. It uses a simple loop to make all the card's buttons show up.

The Buttons

Button 1 is one of the most complex. It's the button that kills the Scouting Toolkit, deleting all its buttons and fields from a card, including itself. Button 1's script handles one kind of message, a *mouseUp*. The button will not delete the ST from the STBC; in that case, it'll flash a warning field at the user. On any other card, though, the message handler removes each button and field via a Victorian clockwork sort of process, as follows.

It turns the cursor into the button tool. For each button it then puts a click at that button's location to select it and calls the HyperCard Edit menu's Clear Button command. Next, it turns the cursor into the field tool. For each field it then puts a click at that field's location and calls on the Edit menu's Clear Field command.

These are two nice HyperCard features: simulating mouse actions and making menu selections. Note how the handler can remove the button whose script it's in. Also, note how I make sure the *userLevel* variable's

(continued from page 85)

high enough to let the handler pull off these tricks and am careful to restore the original *userLevel* when done. (*UserLevel* is a global-state variable that controls access to various levels of HyperCard features.)

Button 14's also interesting. It's the button that replicates the Scouting Toolkit onto a new card. On a mouse click, it copies itself to the Clipboard. Then when you paste it onto a new card, it brings over the entire ST and kills itself. How? Well, once pasted on the replication target, it receives a *newButton* message. The *newButton* handler starts a series of trips, going back to the STBC, copying a button, returning to the target card, and pasting the copied button in. Once all buttons have been transferred, it gets rid of itself, using the technique described earlier.

This process is slow. Using HyperCard's Lockscreen command, so HyperCard doesn't have to draw the screen on each stack round trip, speeds it up somewhat. Still, it'd sure be easier if multiple stacks could be open simultaneously.

Button 3, used to open up the ST, passes a button click on to another button. Button 4 plays music to accompany and emphasize its action. Buttons 5–8 are simple toggle switches. Buttons 9–12 use menu commands. Button 13's analogous to button 2. Button 15 passes a button click on to button 4.

The Fields

There are only two fields. Field 1, normally hidden, shows up and displays a copyright notice under the control of button 15. Field 2, also normally hidden, shows up when someone tries to click button 1 from the STBC. Hidden fields are a neat way to add a sense of animated intelligence to your HyperCard work.

Going Further

If you use the ST for a while, you'll want to tweak it. Two examples that I've worked on are installation into a background and being able to change the spatial configuration of the buttons, with the ST remembering that change. What's interesting is that low levels of object self-manipulation can be done in straight HyperTalk, without resorting to XCMD and XFCN work.

The more I work with HyperTalk/HyperCard, the more I appreciate the convenience, ease, tweakability, and power of the tool. There's a certain simple pleasure to working in its environment, one I haven't had since early BASIC on early micros.

Now, if they can just reduce isolation and modality and get that darned execution speed up. . . .

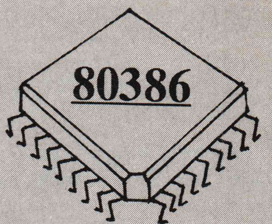
Reader Mail Snapshots

Thanks to all of you who've proffered feedback—oral, electronic, paper—on my first column (January). I appreciate each piece, pro and/or con. It's a major navigational aid. A few quick gleanings:

Wayne Pollock mentioned the gap in Mac literature between books for novices and those for experts and expressed hope that this column might fill part of that hole. I agree,

SERIOUS DEBUGGING AT A REASONABLE PRICE

Soft-ICE™



All the speed and power of a hardware-assisted debugger at a software price - $386

Features

Real-time break points on:
- Memory locations
- Memory ranges
- Execution
- I/O ports
- Interrupts (hardware and software)

Dual monitor support

System memory map

Regain control with a keystroke

Even with the following code:

```
CLI
MOV   AL,0FFH
OUT   21H,AL
JMP   $
```

and much, much more

How Soft-ICE works

Soft-ICE unleashes the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment. Soft-ICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to add real-time hardware-level breakpoints to your existing DOS debugger. To use Soft-ICE you simply pop the Soft-ICE window up through a key sequence, set your hard break points, then return to your soft debugger. As the target program is executing, Soft-ICE recognizes when the breakpoint conditions have been reached and gives control back to your soft debugger. And this is all done at full 80386 speed! Soft-ICE can also be used in stand-alone mode. This comes in handy if you are debugging loadable device drivers, interrupt handlers, or terminate and stay resident programs. All of the standard debugging features are available to help you find the most difficult systems problems.

Benefits

- **Works with CodeView** --To get you up and going as fast as possible, Soft-ICE is designed to work with your existing software debuggers such as CodeView and Periscope I & II.
- **Breaks the 640K barrier** -- If you have extended memory, Soft-ICE takes up **ZERO** bytes of memory in the first 1MB of address space. This means you can load and debug your largest programs.
- **Power of an in-circuit emulator** -- At only $386, you can give every member of your software development team the power of an in-circuit emulator.
- **AT compatible 80386 PC's** -- Soft-ICE works with all AT compatible 80386 PC's, such as COMPAQ's Deskpro 386 and the IBM Model 80.
- **Easy to learn** -- Soft-ICE is so easy to learn that you can be finding bugs with Soft-ICE by the time you could install a hardware-assisted debugger.

"Since Soft-ICE doesn't take up any of my memory I have it in my AUTOEXEC.BAT to load every day... It has saved me at least one month's time on my latest device driver program." Peter Ricker, President of Maverick Software

30 day money-back satisfaction guarantee. Visa and Master Card accepted. Ask about our coupon program.

To order or to get more information, call (603) 888-2386

NU-MEGA TECHNOLOGIES
P.O. BOX 7607
NASHUA, NH 03060-7607

CIRCLE NO. 174 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP

helps save time, money, and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

MacFortran/020 by Absoft. Optimized FORTRAN-77 with VAX, 8X extensions, 68881 support, arrays memory or disk, complex math. Source debug, toolbox, C interfaces. MAC 512, SE, II $449

AI Expert System Dev't

Arity Combination Package	PC	$ 979
Experteach II	PC	$ 339
Level 5 - formerly Insight 2	MS	$ 589
T.I.: PC Easy	PC	$ 435
Personal Consultant Plus	PC	$2589
Turbo Expert-Startup(400 rules)	PC	$ 119
Corporate (4000 rules)	PC	$ 339

AI-Languages

A.P.T. - Active Prolog Tutor	PC	$ 49
ARITY Prolog - Interpreter	PC	$ 229
PC Scheme LISP - by TI	PC	$ 79
TransLISP - learn fast	MS	$ 79
TURBO PROLOG by Borland	PC	$ 69
Turbo Prolog Toolbox	PC	$ 69

C Language-Compilers

AZTEC C86 - Commercial	PC	$ 499
C86 PLUS - by CI	MS	$ 359
Datalight Optimum-C	MS	$ 109
High C Optimizing Compiler	PC	Call
Instant-C - source, debug.	MS	$ 369
Instant-C/16M	PC	Call
Lattice C - from Lattice	MS	$ 259
Microsoft C 5.0 - Codeview		Special *
Microsoft Quick C		Special *
Rex-C/86-standalone ROM	MS	$ 695
Turbo C by Borland	PC	$ 67

C Language-Interpreters

C-terp by Gimpel - full K & R	MS	$ 219
C Trainer - by Catalytix	PC	$ 89
Interactive C by IMPACC Assoc.	PC	$ 189
Run/C Professional	MS	$ 145
Run/C Lite	MS	$ 79
Turbo C-terp	PC	$ 119

C Libraries-Files

BTree Source, no royalties	MS	$ 69
CBTREE-Source, no royalties	MS	$ 129
ctree by Faircom-no royalties	MS	$ 309
ctree w/ rtree	MS	$ 519
rtree - report generation	PC	$ 239
dB2C Files	MS	$ 259
dB2C Toolkit V2.0	MS	$ 249
dbVISTA - Object only		Special *
dBx - translator	MS	$ 299
w/source to library	MS	$ 419

FEATURES

Foxbase PLUS by Fox Software. DBMS conforms to standard MAC interface. Up to 9 definable output windows, definable buttons, import/export hyper-card stacks, MAC, Developer Kit, $339

C Talk by CNS - Object-oriented C programming. Smalltalk-like message formats, contains encapsulation, messaging, inheritance. MS, Lattice, Turbo, C86. MS $ 129

***Mention "Special DD588" and get both a good price and FREE Software!**

FREE Catalogs and Guarantee

Whether you're searching for an obscure product no one seems to know about, or you just want to know which of 5 competitors makes the most well-regarded product, our catalogs make finding software easier.

- Comprehensive Product Listing, filled with over 1,000 products.
- Popular Programmer's Tools, containing the most-requested titles (over 300 in all) from each category.
- Dbase Programmer's Catalog with over 60 development tools.

We'll also help you select products with free advice or literature. Plus full guarantee on any recommended product.

Call to request a catalog or information today.

Our Services:

- International Sales Desk
- Compare Products
- Help find a Publisher
- Evaluation Literature FREE
- Programmer's Update
- Dealers Inquiry
- Newsletter
- Rush Order
- Over 700 products
- National Accounts Center

C Libraries-General

Blackstar C Function Library	PC	$ 99
C Tools Plus - V5.0	PC	$ 99
C Utilities by Essential	PC	$ 119
Entelekon C Function Library	PC	$ 119
Greenleaf Functions	PC	$ 129
LIGHT TOOLS by Blaise	PC	$ 69
Turbo C Tools by Blaise	PC	$ 99

C-Screens, Windows, Graphics

C Power Windows by Entelekon	PC	$ 129
C Worthy Interface Library	PC	$ 249
Curses by Aspen Scientific	PC	$ 109
dBASE Graphics for C	PC	$ 69
ESSENTIAL GRAPHICS-fast	PC	$ 185
FontWINDOW/PLUS	PC	$ 225
GraphiC - new color version	PC	$ 279
Greenleaf Data Windows	PC	$ 155
w/source	PC	$ 259
LightWINDOWS/C for Datalight	PC	$ 79
Panel/TC - for Turbo C	PC	$ 95
PC Forms-by Golden Software	MS	$ 109
ScreenStar - with source	PC	$ 169
Terminal Mapping System	PC	$ 279
TurboWINDOW/C-for Turbo C	PC	$ 75
View Manager - by Blaise	PC	$ 199
Vitamin C - source, menus	PC	$ 159
VC Screen - screen paint	PC	$ 79
Windows for C - fast	PC	Call
Windows for Data - validation	PC	Call

DataBase & File Management

Advanced Revelation	PC	$ 779
CQL	PC	$ 359
DataFlex by Data Access	PC	$ 595
DataFlex multiuser	PC	$1049
Magic PC	PC	$ 169
Paradox - original	PC	$ 369
Paradox V2.0 List: $725	PC	$ 499

DBASE Language

Clipper compiler	PC	$ 399
dBASE III Plus	PC	$ 399
dBASE III LANPack	PC	$ 649
DBXL Interpreter	PC	$ 99
FoxBASE+ - V2.0	MS	$ 259
Quicksilver Diamond	PC	$ 369

DBASE Support

dAnalyst	PC	$ 219
dBASE Tools for C	PC	$ 65
dBRIEF with BRIEF-Auto-Indent, views structures	PC	Call

RECENT DISCOVERY

FoxBASE 386 by Fox Software. Single-user development package. Interactive dot-prompt mode, FOXBIND utility, on-line help, full manual, pseudo-compiler. Address up to 16M RAM. PC $519

DBASE Support cont.

dBC III Plus	PC	$509
dbug III	MS	$179
Dialect UI	PC	$ 45
dFLOW - flow charts	MS	$125
Documentor - dFlow superset	MS	$229
Genifer by Bytel-code generator	MS	$279
QuickCode III Plus	MS	$189
R&R Report Writer	MS	$139
Seek-It - Query-by-example	PC	$ 79
Silver Comm Library	MS	$139
Tom Rettig's Library	PC	$ 79
UI Programmer-user interfaces	PC	$249

Debuggers

Breakout - by Essential	PC	$ 89
CODESMITH - visual	PC	$ 99
C SPRITE - data structures	PC	$119
DSD87	PC	$ 79
Periscope I	PC	$275
Periscope II	PC	$139
Periscope II-X	PC	$105
Periscope III-10 MHZ Version	PC	$795
Pfix-86 Plus - by Phoenix	PC	$209
SoftProbe II - embedded systems	PC	$695

Editors for Programming

BRIEF Programmer's Editor	PC	Call
de - EMACS-style	PC	$ 65
EMACS by UniPress Source: $895		$265
Epsilon - like EMACS	PC	$149
KEDIT - like XEDIT	PC	$ 99
ME Macro Editor - Source	PC	$ 79
MKS VI	MS	$ 65
PC/VI - by Custom Software	MS	$109
Personal REXX - V1.6	PC	$ 99
SPF/PC - Version 2.0	PC	$179
Vedit PLUS	MS	$129

Fortran & Supporting

ACS Time Series	MS	$399
Forlib+	PC	$ 55
I/O Pro - includes No Limit	PC	$229
MACFortran by Microsoft		Special *
MS Fortran - 4.0, full 77'		Special *
PC-Fortran Tools - xref, pprint	PC	$165
RM/Fortran	MS	$399
Scientific Subroutines - Matrix	MS	$129

Multilanguage Support

BTRIEVE ISAM	MS	$185
BTRIEVE/N - multiuser	MS	$455
GSS Graphics Dev't Toolkit	PC	$375
Halo Development Package	MS	$389
HALO Graphics	PC	$209
Help/Control - on line help	PC	$ 99
Hoops 3D Graphics Library	PC	$469
Informix 4GL-application builder	PC	Call
Informix SQL - ANSI standard	PC	Call
Instant Programmer's Help	MS	$ 79

Note: Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/per normal item. All prices subject to change without notice.

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

CIRCLE NO. 197 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees, and every product for Microcomputer Programming.

Your Source for Debuggers

Embedded code, C, Asm. Whatever you write, chances are it doesn't run right the first time.

For clear windows on your code, consider these professional tools. Recover from even frozen machines, debug at high level, and keep interactivity in the debugging process.

Call one of our Tech Reps for help choosing TODAY.

Order before 5/31/88 and mention "DD588" for these Special Prices:

	List	Normal	SPECIAL
C Sprite by Lattice	$ 175	$119	**$ 99**
DBug III - for dBASE	$ 195	$179	**$159**
Periscope I-incl. board	$ 345	$275	**$255**
Periscope II-incl. switch	$ 175	$139	**$119**
Periscope III-10 MHZ version	$1095	$795	**$749**
SoftProbe II/TX-Rom	$ 750	$695	**$599**
TURBOsmith-Turbo Pas	$ 99	$ 79	**$ 59**

Multilanguage Support cont.

NET-TOOLS - NET-BIOS	PC	$129
Norton Guides	PC	$ 75
Opt Tech Sort - sort, merge	MS	$ 99
PANEL Plus	MS	$395
Pfinish - by Phoenix	MS	$209
Report Option - for Xtrieve	MS	$109
Screen Sculptor	PC	$ 89
SPSS/PC Plus	PC	$749
Synergy - create user interfaces	MS	$329
XQL - SQL for Btrieve	MS	$649
Xtrieve - organize database	MS	$179
ZAP Communications - VT 100	PC	$ 89

OS Support

DOS Merge 286	PC	$139
Microsoft Windows		Special *
Development Kit		Special *
MKS AWK	MS	$ 65
MKS Toolkit - Unix vi, awk	PC	$115
Norton Utilities Advanced	MS	$ 99
System V/AT Combination	PC	$489
Xtree - Professional	MS	$109

Translator

dB2C - requires toolkit	MS	$ 249
RTC PLUS by Cobalt Blue	MS	$ 399
SofTRAN - Translation Lang.	PC	$ 349
TP2C	PC	$ 199
Turbo-to-C-Tools by TGL	PC	$ 479

Other Languages

ACTOR	PC	$ 419
Ada Dev's Toolkit-Vol.1 & 2	PC	$ 889
Alsys Ada w/ 4 M RAM	PC	$2995
APL*PLUS/PC	PC	$ 439
CCS Mumps - Multiuser	PC	$ 359
Microsoft MASM		Special *
Modula-2 - V3.0 Dev. System	PC	$ 199
Smalltalk/V	MS	$ 85
SNOBOL4+ - great for strings	MS	$ 80
RPG II Compiler by Lattice	PC	$ 639

Other Products

ASMLIB - 170+ routines	PC	$ 125
Back-It by Gazelle	MS	$ 119
Baler	PC	$ 459
Dan Bricklin's Demo II	PC	$ 169

RECENT DISCOVERY

Halo '88 - Update includes scanner control/scanned image manipulation, 144 devices supported, PS/2. Virtual Raster Interface for EMS. New languages Turbo C, BASIC, MS QuickBasic. PC $229

Other Products cont.

Disk Technician-smart upkeep	PC	$ 89
Fast Back Plus	PC	$149
Flash-Up	PC	$ 69
Interactive Easy Flow V5.0	PC	$125
Link & Locate - Intel tools	MS	$309
Mace Utilities	MS	$ 85
MKS Trilogy	MS	$ 99
Plink 86 Plus - overlays	MS	$275
PC-Metric - analyze complexity	MS	$ 89
PVCS by Polytron		Special *
R-DOC/X	MS	$135
risC by IMSI - H.A.L.	MS	$ 75
Sapiens Make	MS	$155
Show Partner F/X	PC	$328
Source Print - V3.0	PC	$ 75
TLIB	PC	$ 89
Tree Diagrammer	PC	$ 65
Visible Computer: 8088	PC	$ 65
WKS Library by Tenon	PC	$ 79

Xenix Unix

C-Terp by Gimpel Software	$379
Cobol - by Microsoft	Special *
Fortran or Pascal-by Microsoft	Special *
FoxBASE+	$649
RM/Cobol	$959
Xenix Complete System 286	$999

SCREEN GENERATOR - WINDOW MANAGER

POPSCREEN™

FOR PROFESSIONAL LOOKING PROGRAMS

Powerful But Simple Display Editor
Makes easy work of creating complex & colorful displays, pop-up menus, windows. Features easy boxes, outlines, block moves, superimpose, complete color & character graphics control, boilerplating, on line help; & much more.

Displays Integrate With Your Compiler's Output
PopScreen outputs your displays to library modules for your compiler. Displays link into your program at link time. No frustrating resident loaders, no loading screens from files on disk. Your displays are part of your program!

Instant Screen Access
Your displays pop to the screen instantaneously - in 1/25th of a second! Open and close windows with a simple procedure call. Your displays are fast, professional, and optimally compressed for storage inside your program.

PopScreen 3.0 Supports:
C: Microsoft, Lattice, Turbo, IBM
8086: will output assembly code
PASCAL: IBM, Microsoft
TURBO PASCAL: inline code
QUICKBASIC: library modules
DBASE III+ loadable .bin files

PopScreen
only $89.00
PS Price: $79

BaySoft Box 6562-P, Albany, Cal. 94706
415-527-3300

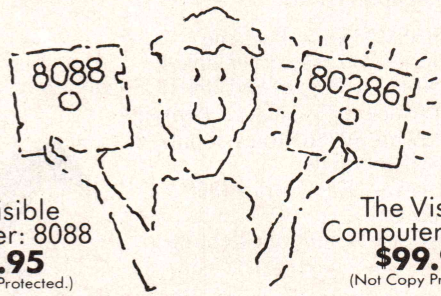
60 DAY SATISFACTION GUARANTEE

CIRCLE NO. 198 ON READER SERVICE CARD

We've Added a New Teacher to Our Assembly Language Program.

Introducing The Visible Computer: 80286–an advanced version of the "Award Winning" TVC: 8088. Its animated simulation of the PC's microprocessor shows you exactly how assembly language instructions are carried out. And it comes complete with a comprehensive 400 p. manual.

TVC lets you choose from 45 demonstration programs that can be executed with the simulator–from simple register loads to advanced programs that manipulate interrupts and perform file I/O. It makes learning assembly language . . . elementary!



The Visible Computer: 8088
$79.95
(Not Copy Protected.)

The Visible Computer: 80286
$99.95
(Not Copy Protected.)

Software Masters™

The Visible Computer: 8088 for IBM PC/XT/AT and true compatibles; TVC: 80286 for IBM AT. To order direct: Software Masters, 2714 Finfeather, Bryan, TX 77801; (409) 822-9490. Include $3.00 handling. Bank cards accepted.

CIRCLE NO. 199 ON READER SERVICE CARD

Call for a catalog, literature, advice, and service you can trust.

HOURS: 8:30 AM. - 8:00 P.M. E.S.T.
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-740-2510

800-421-8006

THE PROGRAMMER'S SHOP
Your complete source for software, services and answers

TO THE MACS
(continued from page 86)

and I'll try. And Scott Knaster's new book, mentioned earlier, should be helpful. Wayne also wants more skeleton code and examples. Good news on that front from Apple itself, where the Developer Technical Support team has started work on a major new code examples effort. And this column will sport a small Multifinder skeleton Real Soon Now, as Prof. Pournelle would say. Wayne also wants some C++ material. Well, as mentioned earlier, I've finally begun fiddling with MPW, partially because I sniff that a Mac C++ will show up in that environment first. When it does, I'll muck about and report back.

James Savidge wrote requesting a little more information about the solitary useless Mac programming tool I mentioned in my first column. It was a C compiler, not one you see advertised anymore, but beyond that I say nothing. I refuse to give a company that won't refund money to a completely dissatisfied customer any form of energy. Besides, I haven't heard or seen squat about the hucksters for years, so hopefully their tent's collapsed and they've sunk back into the dark.

Several programmer buddies were happy to see more Mac coverage here in *DDJ*. I agree. But my favorite piece of mail came from a guy whose name I don't know—the missive's enshrined somewhere down at *DDJ* worldwide headquarters—who was P.O.'d at all the graphics that accompanied the January article. He reads *DDJ* for code, not cartoons. Now *that's* the *DDJ* audience I love! Keep it coming.

Updates and Fixes

Apple's excellent *Human Interface Guidelines* book that I mentioned in the March column is now out in trade form from Addison-Wesley. A lot of the Apple docs will make it out via that route: drafts through APDA, finished stuff via A-W. It's called the *Apple Technical Library*. The folks from Reading do a nice production and finish job.

SuperMac Technology's stopped manufacturing the Enhance board I mentioned in the January column.

Too bad, it was/is a fine product. But we've got the Levco stuff now. Also, the final fix for the parasitic clip problem I mentioned regarding my Enhance board: a piece of carefully connived corrugated cardboard artfully wedged twixt the daughterboard and the clip, applying that constant pressure so needed for reliable electromechanical connection—Kludge Klassic #456.

Minor (Ha! Aren't they all when you own 'em?) bug in the March column's source code: the second instruction in the *doDrawCntl* routine, shown on page 70, should branch to that routine's closing *RTS*, not to the label drawn. So put a label on the *RTS*, say *DoDrCnBye*, and change the line to:

```
BEQ doDrCnBye ; it's invisible, so
no need to draw it
```

This is a branch rarely taken, and the flawed version has a solid chance of not flaming a program. Classic formula for insect survival.

Shut Up and Wrap-Up

Yow, it's time to recede. Thanks to all the nice Maccites at the Expo who took time to share thoughts and endure my nonstandard interviewing techniques, including (but not limited to, as I'm prone to quirky memory and misplaced interview tapes) Harvey Alcabes, Scott Boyd, Frank Catalano, John Draper, Joseph Edozien, Chris Espinoza, Amy Goldsmith, Michael Gosney, Michael Green, Ray Heizer, Glenn Hoffman, David Intersimone, Steve Jasik, Margie Kaptanoglu, Bill Kelly, Scott Kim, Scott Knaster, Richard Koch, David Krathwohl, Lance Lewis, Ken Loomis, Greg Marriot, Julia Menapace, Jeff Nutt, Peter Olson, Howard Pearlmutter, J. Scott Phillips, Heidi Roizen, Gerard Schutten, David E. Smith, Joel Spiegel, Michael Swaine, Wes Thomas, Randall Tinkerman, Neal Trautman, and Steve Splonskowski. With special thanks to Amos Gottlieb and roommates for the fine and classic Haight Street gestalt accommodations.

Next time out? I've given up the prediction racket. My batting average is invisible. But, hey, it'll be fun no matter what, eh? Software! Books! People! Code! Especially code, because I've taken it light this month. So dive happily into those fine mind exercisers y'all are so addicted to, and come back ready to do the logic boogie.

Bibliography

Apple Computer Inc. *HyperCard Developer's Toolkit*, Version 1.0. Available through APDA. Part #KMS036.

Apple Computer Inc. *HyperCard Script Language Guide*. Available through APDA. Part #KMB009.

Knaster, Scott. *Macintosh Programming Secrets*. Reading, Mass.: Addison-Wesley, 1987. ISBN 0-201-06661-0.

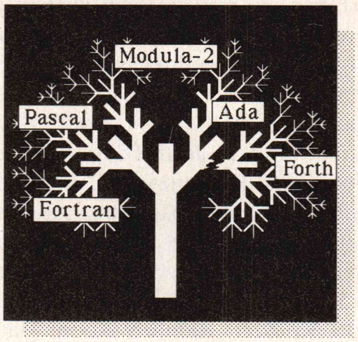
DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

Vendor

APDA
290 S.W. 43rd St.
Renton, WA 98055
206-251-6048

Convincing Pascal to Read Non-Pascal Files



Sometimes a feature of a language is merely a defect put in a favorable light. It all depends on what you're trying to accomplish. Pascal, for example, insists that all files be bound to a data or record type: very noble from the standpoint of preserving the purity of strong typing but often an obstacle when trying to process files formatted in languages other than Pascal.

Specifically, the kinds of files I'm talking about are self-describing tables such as those generated by dBASE, Reflex, and other database programs. Typically, such files begin with several data structures describing the contents, followed by any number of fixed-format data records. It's easy to mix record types with C and assembly language and even BASIC, all of which support free-form files. You have to convince Pascal to do it, though, and the trickery for doing so is the subject of this month's column.

I'll also respond to a reader's complaints about Turbo Pascal 4.0.

Creating a Table File

Rather than covering a specific vendor's table format, I've developed a simple model for this article that is typical of these files in general. What makes it simple is not the file structure itself but instead the number of options. Data records can consist of only two field types: integers and character arrays. The thrust here is the principles, and there's no sense muddying the waters with a num-

by Kent Porter

ber of options that you can figure out for yourself.

The typical table file begins with a fixed-length preamble (256 bytes in this case) containing a header record and field descriptors. The header record is a fixed structure that contains several fields giving basic information about the contents. In this case, the header record is 24 bytes long and contains the fields shown in Table 1, page 95.

Signature is an invariant value written at a fixed place to identify the file as belonging to the application. If some other value appears in that position, the file doesn't follow the rules given here and can't be processed. The value of *signature* for this application is 19364 (4BA4h), and it appears in the first 2 bytes of the file.

The *nrecs* field tells you how many data records the file contains. *Tablename* is a packed array of ten characters giving the name of the table; not all vendors have an analogous field in the header record. *Datastart* expresses an offset, with respect to the start of the file, to the first data record. This is a *long* (32 bits on a PC) integer to correspond with the usage of *fseek/ftell* in C. Turbo Pascal 4.0 and Microsoft Pascal similarly use a *long* integer for their *SEEK* procedures. The last two fields, *descrsize* and *ndescr*, have to do with the next part of the file preamble.

A data record consists of one or more fields, each of which has three attributes: a name, a data type, and a length. These constitute a field descriptor, which has the form shown in Table 2, page 95. Each data field has one descriptor, hence there are *ndescr* field descriptors of *descrsize* following the header record. In the programs that accompany this article, for example, there are two fields (*ndescr*=2), so there are two descriptors.

Thus the preamble consists of a fixed header record followed by a variable number of descriptor records, each of which has a fixed format. Taken together, they describe the data content of the file. The unused portion of the preamble is filled with uninitialized garbage.

The data itself begins at byte offset *header.datastart* from the beginning of the file. Each record corresponds to a row in the data table and each field to a column. The descriptor records describe the columns, and the length of any given record is the sum of all *flen* fields in the descriptors. There are *header.nrecs* records. The file is thus a self-describing entity, and the program's job is to interpret the descriptions in order to extract the data. Figure 1, page 95, shows the format of a simple table.

Listing One, page 69, is a generic C program (MKTABLE.C) that creates a table with the preamble described here. The program then requests data entry and writes out the data records you type in response, saving them in a file called database.xyz. In order to build an adequate table, enter at least three or four records. Terminate data entry by pressing Return when the program asks for a name. The program then updates the header record to reflect the number of data records entered and closes the file. This is a vastly simplified version of a database management package, but it generates a complete table of the same sort that flows out of dBASE and other table-oriented database products.

Translating Strings

The references to *pac* (a packed array of characters) in Figure 1 and Table 1 point up a fundamental difference between Pascal and lower-level languages such as C and assembly language. Although not de-

Why We're Betting a Million Lines of Code on the SAS/C™ Compiler.

At SAS Institute Inc., we've invested more than 10 years of research—and over a million lines of code—in the SAS® System, the world's leading data analysis software. So you can bet we left nothing to chance when we chose the C language for the next generation of our software.

We selected C for the portability it would bring to the SAS System, but weren't about to risk our code on just **any** mainframe C compiler. So we tried them all. When none could meet our exacting requirements, we created our own: the SAS/C compiler.

We Developed It. Support It. Use It.

The SAS/C compiler set new standards for efficiency and technical quality, with:

- A source-level debugger that includes structure display, ABEND recovery, and debugger I/O exits for debugging specialized applications
- Reentrant object code
- Highly optimized generated code
- Use of standard IBM linkage conventions, with support for 31-bit addressing
- A CMS Rexx/TSO CLIST interface
- Support for signal handling including program checks and terminal interrupts, and non-standard signals such as timer interrupts and stack overflow
- Many built-in functions including string handling
- In-line assembler.

And when we combined these features with outstanding technical support and frequent updates—both provided free—software developers everywhere took notice. The SAS/C compiler is now the market leader, installed in hundreds of commercial firms and academic institutions.

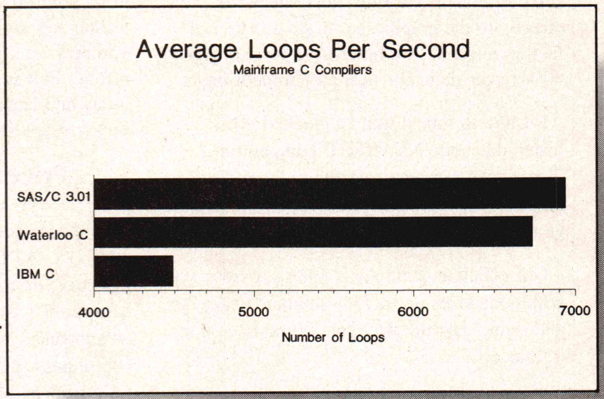
Test It. Compare It. FREE for 30 Days.

We're betting you've set the same high standards. That's why we'd like to send you the SAS/C compiler, under OS or CMS, for a free 30-day evaluation. We'll also send you a free copy of a leading benchmark program. Compare our compiler with any other. Odds are, you'll choose the SAS/C compiler.

Just mail the coupon below. Or call your Software Sales Representative at (919) 467-8000.

SAS Institute Inc.
SAS Circle ☐ Box 8000
Cary, NC 27512-8000
Phone (919) 467-8000
Fax (919) 469-3737

Using a C version of the Dhrystone benchmark, the latest SAS/C compiler release produces the fastest code among the top 3 mainframe compilers. It even tops our own previous release by 35%.



I'd like to put the SAS/C™ compiler to the test with a free 30-day trial, and my free copy of the Dhrystone benchmark program. Give me the details.

Please complete, or attach your business card.

Name ______ Title ______

Company ______

Address ______

City ______ State ______ ZIP ______

Telephone ______

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA, 27512-8000

DDJ 5/88

STRUCTURED PROGRAMMING
(continued from page 92)

fined in the academic standard, the *pac* string type is supported by most real-world Pascal compilers. It's really a stretched *PAC*, the difference being that element 0 contains the string length, with the first valid character at element 1. Thus, at the data level, a string containing the word *Pascal* looks something like *6Pascal*. If the string were declared as:

```
VAR strng : STRING [10];
```

the last 4 bytes would contain garbage. The compiler inserts string-handling routines that pay attention to the length byte.

C handles string data differently, and most assembly-language programmers use the same convention as C. It's so common, in fact, that it has a name: ASCIIZ. In ASCIIZ, the 0th element contains the first character of the string. There is no length indicator; instead, end-of-string is signified by ASCII value 0, or *CHR (0)* in Pascal notation. This is merely a packed array of characters with a special end sentinel. The C term for it is a *null-terminated* string.

The *asciiz* function in Listing Two, page 70, transforms ASCIIZ strings into Pascal strings. Because it's possible that a string might exceed its maximum length, the function also takes the *max* parameter, which limits the number of characters in the result; it's either *max* characters or everything up to the null terminator, whichever comes first.

This string-translation routine and the liberal use of variant records are two keys to convincing Pascal to read non-Pascal tables. The third is processing the file on a byte-by-byte basis. Here's how it works.

Processing the File

The program declares the table as *FILE OF BYTE* and opens it. The first step reads the 24-byte header record into the stream variant of the *headrec* structured variable. This is necessary because the file is of type *BYTE*; all file reads are done in the same way. Access to the data elements is via the other variants, as in the next step, which checks *signa-*

ture. Execution continues if the *signature* is correct and ends with a message otherwise.

Procedure *showHeaderInfo* lists information from the header record. Note that the table name is drawn from the second variant of the *headrec* record. Why? Because the *asciiz* function expects an argument of type *pac*, which is 20 bytes long, whereas the real *tablename* field is only 10 characters long. This is a trick to prevent the compiler from choking on a mismatched type.

The *getDescriptors* procedure, called by *showHeaderInfo*, reads the field descriptors that follow the header file. The program assumes a maximum of ten fields for the table when it declares the *field* variable, which is an array of *fieldrec* structures. The *header.ndescr* variable governs the actual number read from the file. *ShowHeaderInfo* uses the descriptors to display information about the fields. The *showData* procedure uses them more extensively.

Before calling *showData*, however, the program first calls the Pascal *SEEK* procedure. The purpose is to reposition the file pointer to the start of the data records, which is past the unused portion of the preamble. *ShowData* can now read and process the table's data contents sequentially.

A couple of local variant record types provide the means for fetching integers and ASCIIZ strings. Again, the stream component furnishes type compatibility with the file. Because the host processor and not the compiler establishes the format of the integer type, an integer taken from the file as two consecutive bytes can be plucked directly from the variant without translation. The character field is accessible via a call to *asciiz*.

A pair of nested loops control the reading and display of data fields. The outer loop repeats for the number of records in the file, as given by *header.nrec*. The inner loop processes individual fields, stepping through the array of descriptors in order to determine what to read next from the file. Because it's loop-driven, *showData* can process any

Name	Type
signature	word
nrecs	word
tablename	pac [10]
datastart	longint
descrsize	integer
ndescr	integer

Table 1: *Names and types of fields in the header record*

Name	Type
fname	pac [20]
ftype	integer
flen	integer

Table 2: *A data record's field descriptors*

Header	$4BA4	(=signature)
	4	(=nrecs)
	Age list	(=tablename)
	256	(=datastart)
	24	(=descrsize)
	2	(=ndescr)
Descriptor#1	NAME	(=fname)
	1	(=ftype {pac [20]})
	20	(=flen)
Descriptor#2	AGE	(=fname)
	0	(=ftype {integer})
	2	(=flen)
Rest of preamble	(garbage filler)	
Data records (datastart)	Ken Barker, 46	
	Tim Madden, 38	
	John Joyner, 42	
	Jim Hull, 59	

Figure 1: *Format of a simple table*

36% Savings

Yes! I want to subscribe to

MicroSystems

JOURNAL for the PC Systems Integrator

and save over 36% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ______________________ Exp. date __________

Signature ______________________

Name ______________________

Address ______________________

City ______________________ State ________ Zip ________

*Savings based on the full one-year newsstand rate of $47.70. All foreign countries please add $7 per year for surface mail; Canada & Mexico add $17 per year for airmail; other countries add $28 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

213S

A PUBLICATION OF M&T PUBLISHING, INC.

36% Savings

Yes! I want to subscribe to

MicroSystems

JOURNAL for the PC Systems Integrator

and save over 36% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ______________________ Exp. date __________

Signature ______________________

Name ______________________

Address ______________________

City ______________________ State ________ Zip ________

*Savings based on the full one-year newsstand rate of $47.70. All foreign countries please add $7 per year for surface mail; Canada & Mexico add $17 per year for airmail; other countries add $28 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

213S

A PUBLICATION OF M&T PUBLISHING, INC.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

MicroSystems
JOURNAL for the PC Systems Integrator

Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

MicroSystems
JOURNAL for the PC Systems Integrator

Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

MicroSystems
JOURNAL for the PC Systems Integrator

Box 3713
Escondidio, CA 92025-9843

(continued from page 95)

number of records consisting of any number of integer and ASCIIZ fields in any order without modification. Additional data types would require the appropriate structure definitions and expansion of the *CASE* statement.

This is not a complete table system, of course, because it lacks date, floating-point, and Boolean types. Also, it's not compatible with any existing database package's file format. Given the specifications for a file and the techniques presented here, however, you should be able to write a Pascal program that reads non-Pascal files with header records.

Turbo Pascal 4.0 Flames

The mail the other day brought a letter from Charles Linett, who heads up the Computer Science Staff at the Census Bureau. Charles and his folks use Turbo Pascal for communications programs of 15,000+ lines, and he's not amused by Version 4.0. Here's part of what he has to say:

"I see two rather large defects bordering on the semicalamitous for our type of work.

1. There are no overlays (as there were in Version 3.0). Borland has solved that problem in two suave ways, however. First, the company told us that if we used overlays, then we needed only to rewrite our programs (thanks, fellas, you're a big help). Second, Borland found the part of the documentation least likely to be read (a file called Q&A) and wrote that it recognized the need and was working on something 'intelligent.' I can only hope it gets it done before those awaiting the feature die of old age.

2. The manual is awful and (worse yet) has almost no chance of improvement. It requires so many additions and corrections that what is called for is a new manual altogether. If 20 pages need changing in a real manual, you send the customer those 20 pages and let him or her stick them in the book. This cannot be done with the 4.0 manual because it's glued together in one big lump."

No quarrel with the first point, Charles; I'll get back to it in a minute. As for the second, probably a lot of us aren't crazy about a bound book as a manual. And that one in particular is too thick; you can't spread it out on the desk for reference without either breaking the spine or putting barbells on it for paperweights. But the adjective *awful* is kinda harsh. Versions 2.0 and 3.0 had bound manuals, too, and Borland isn't the only company whose docs come this way.

Nobody's ever told me this, but I suspect the purpose of bound docs is to discourage pirates from photocopying the manual. Maybe if the world was a more honest place, vendors such as Borland wouldn't resort to defensive tactics. Piracy is just another name for theft.

Yeah, I don't like the manual either, but it's a whole bunch better than its predecessor in terms of both quality and content. And manual corrections conveyed via

STRUCTURED PROGRAMMING

(continued from page 97)

READ.ME files are hardly a Philippe Kahn innovation.

Now for the overlay fiasco. No doubt about it, Borland shot itself in the foot by dropping overlays. Probably it figured it could get away with it because Version 4.0's .EXE files break through the infamous 64K barrier of Version 3.0's .COM files. Somebody should have surveyed the user community before Borland yanked the rug from under it.

But there's an alternative for Charles and anybody else who got abandoned. It's a product called Overlay Manager 4.0 from TurboPower Software (3109 Scotts Valley Dr., Ste. 122, Scotts Valley, CA 95066; 408-438-8608). Costing $45, this is an interactive program that lets you break a compiled .EXE file of any size (up to about 1 Mbyte) into any number of overlays. For truly enormous programs, there's another utility in the package that effects chaining. The slim 30-page manual is excellent and so's the quality of the software; TurboPower produces good stuff. Highly recommended if you need overlays.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call 415-366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

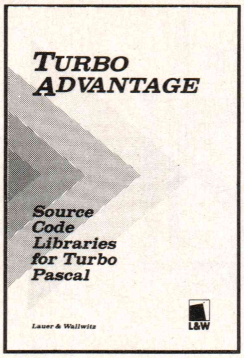
(Listings begin on page 69.)

Vote for your favorite feature/article.
Circle Reader Service No. 7.

All Source Code Maximizes Turbo Pascal 4.0

TURBO Advantage: Source Code Libraries for Turbo Pascal

by Lauer & Wallwitz



The TURBO **Advantage** library contains more than 220 routines complete with source code, sample programs and documentation. Routines are organized and documented under the following categories:

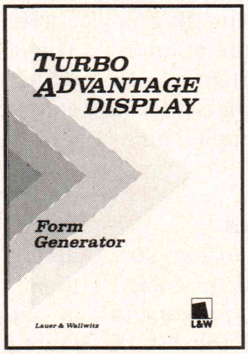
• file management • MS-DOS support • sorting routines and spline integration • string operations • arithmetic calculations • data compression • differential equations • Fourier analysis and synthesis • matrices and vectors • statistics • and more!

A detailed manual includes a description of each routine, an explanation of the methods used, the calling sequence, and a simple example. Source code is included. For MS-DOS systems.

Manual & Disk (MS-DOS) Item #26-7 $29.95

TURBO Advantage Display: Form Generator for Turbo Pascal

by Lauer & Wallwitz



The **TURBO Advantage Display** form generator makes it easy to design and process forms to fit your needs. The package includes an easy-to-use form processor, 30 Turbo Pascal procedures and fuctions to facilitate linking created forms to your program, full source code, and documentation. For MS-DOS systems. Some of the **TURBO Advantage** routines are necessary to compile T**URBO Advantage Display.**

Manual & Disk (MS-DOS) Item #28-3 $39.95

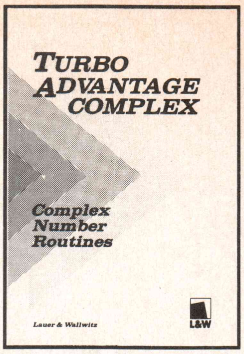
SPECIAL OFFER

Receive TURBO Advantage together with TURBO Advantage Complex or TURBO Advantage Display and SAVE 20%

TURBO Advantage & TURBO Advantage Complex
Item #070A $55
TURBO Advantage & TURBO Advantage Display
Item #070B $55

TURBO Advantage Complex: Complex Number Routines for Turbo Pascal

by Lauer & Wallwitz



TURBO Advantage Complex provides procedures for performing all the arithmetic operations and necessary real functions with complex numbers. Each procedure is based on predefined constants and types. By using these declarations, the size of arrays are easily adapted. Each type declaration is a record with both a real and an imaginary part. Use these procedures to build more sophisticated functions in your own programs.

TURBO Advantage Complex also demonstrates the use of these procedures in routines for vector and matrix calculation with complex numbers and variables; simultaneous Fourier transforms; calculations of convolution and correlation functions; low-pass, high-pass, band-pass, and band-rejection digital filters; and solving linear boundary-value problems.

Source code and documentation included. For MS/PC-DOS systems. Some of the **TURBO Advantage Complex** routines are most effectively used with routines contained in **TURBO Advantage.**

Manual & Disk (MS-DOS) Item #27-5 $39.95

TO ORDER: Return this order form, along with your payment to M&T Books, 501 Galveston Drive, Redwood City, CA 94063. Or CALL TOLL FREE 800-533-4372, Mon. - Fri. 8 a.m. - 5 p.m. Pacific Standard Time. In CA CALL 800-356-2002.

Name ______________________________

Address ______________________________
(Please use street address, not P.O. Box)

City ______________ State _____ Zip __________

Day Phone ______________________________
(In case we have questions about your order.)

☐ Check enclosed. Make payable to M&T Books.

☐ Charge my: ☐ Visa ☐ Master Card ☐ AmEx

Card # ______________ Exp. Date __________

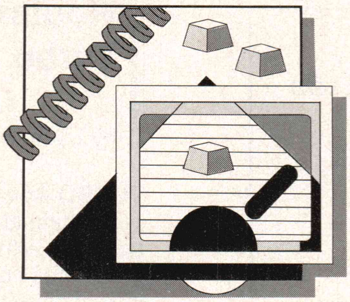
Signature ______________________________

Qty	Item #	Description	Price

Subtotal __________
Sales Tax - CA residents must add __________
applicable sales tax ___ % __________
Shipping and Handling ($2.95 per book) __________
Total __________

3139A

Parallel Processing, Object-Oriented Programming, and a Reading List



Several of the key difficulties of my original text cluster about the concept of a paradigm...One sympathetic reader...prepared a partial analytic index and concluded that the term is used in at least twenty-two different ways.

—*Thomas Kuhn*

Authors who use the word *paradigm* seem to permit themselves to mean by it a large number of different, and perhaps contradictory, things; and as I am committing myself to dealing with paradigms each month, I will honor this wise and liberal tradition.

A column called Programming Paradigms should, though, at least, deal with the issues Robert Floyd brought up in his 1978 Turing Award Lecture, "The Paradigms of Programming" (even though Floyd didn't define paradigm either). Floyd deplored the segmentation of computer science into narrow communities, "each speaking its own language and using its own paradigms...well-defined schools of LISP programming, APL programming, ALGOL programming, and so on." Such communities develop not just around languages but around any broadly applicable problem-solving technique, including such diverse techniques as recursion, stepwise refinement, generate-and-test, and data-flow design.

Programmers advance in their careers by learning a community's shared techniques, terminology, values, prejudices, model problems,

by Michael Swaine

and concrete examples of how to solve such problems—what Thomas Kuhn, who investigated scientific paradigms, calls puzzle solving. This is the process of adopting a paradigm, and it prepares the programmer to answer an ad for a "C programmer" or a "software engineer" or a "knowledge engineer" or a programmer "with experience in object-oriented design."

A column called Programming Paradigms should offer some insight to the atypical programmer for whom this is not enough, the programmer who wants to add to his or her repertoire of paradigms.

This column will attempt to do that by exploring techniques from parallel processing, programming in logic, functional programming, object-oriented programming, and other alternative models. The focus will be on pointing out the alternatives to familiar, conventional methods. I can't think of a better place to begin the exploration than in the wilderness of MIMD parallel processing.

Parallel Paradigms

When I conducted an informal survey through the pages of the magazine last year, parallel processing came out at the top of the list of topics readers would like to see *DDJ* cover more often. It's not surprising; parallel processing promises increased throughput independent of other speedup techniques, and parallel processing raises new problems to be explored—a lot of them—the parallel world is bigger than the sequential world. There are many paradigms of parallelism, some better mapped than others. One of the least explored is MIMD parallelism.

The term *MIMD* means multiple instruction, multiple data. Logically, there are four such terms, representing the application of parallelism to instructions, data, neither, or both. The terms can be applied to both architectures and algorithms. SISD (single instruction, single data) is the familiar sequential Von Neumann model for computer architecture and programming; and MISD, in which multiple instructions are applied to single data items, turns out to be in practice indistinguishable from SISD. That leaves two broad paradigms of parallel processing: single instruction, multiple data (SIMD) and multiple instruction, multiple data (MIMD).

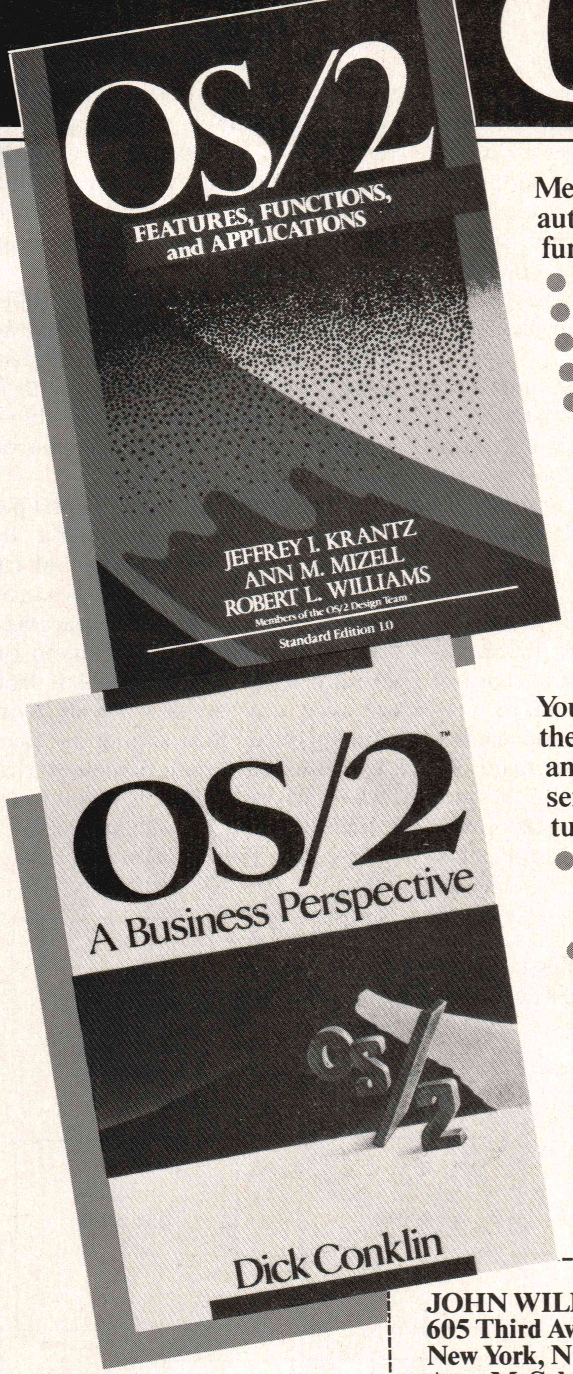
SIMD breaks down into two lower-level paradigms, each with its community of practitioners, traditions, model problems, and typical solutions. These are array processing and vector processing.

The array-processing paradigm (see Figure 1, page 102) involves a sequence of instructions applied concurrently to disjoint sets of data. The ICL Distributed Array Processor, which consists of a control unit and a 64×64 array of processors, each with its own local memory, is one example of this paradigm. In it, the control unit drives the parallel processors, which all perform the same operation in lockstep on the data in their local memory. This paradigm is natural for matrix mathematics, graphics processing, and other uniform operations on arrays of data—hence the name array processing.

The vector-processing paradigm, which is also called pipelining, involves instructions overlapped on disjoint sets of data (see Figure 2, page 103). With pipelining, additional operands can be fed into an operation before it has finished with the first because the operations are broken down into stages and the stages are processed in parallel.

You can see architecture designed for pipelining in supercomputers such as the Cray-1, which are often called vector processors because they process a vector of operands in

THE AUTHORITY ON OS/2...



Members of the OS/2 design team bring you the authoritative guide to OS/2's powerful features and functions:

- OS/2 memory and multitasking facilities
- OS/2 dynamic linking and I/O capabilities
- Interrupt-driven device management
- Interprocess communications
- C programming examples that describe OS/2 application programming interfaces down to a functional level

OS/2™ — $24.95
Features, Functions, and Applications

You need a business perspective to help you compare the advantages of OS/2 over DOS. Now Dick Conklin an IBM insider who develops and presents OS/2 seminars to dealers nationwide, explains all the features and functions of OS/2, including:

- How OS/2 outperforms DOS in every way—from memory, graphics, and multitasking, to business applications
- How to install, customize, and manage OS/2 to meet your business needs
- How to convert from DOS to OS/2 and migrate your existing programs to run on it

OS/2™ — $21.95
A Business Perspective

At bookstores or use this coupon to order

WILEY
Business/Law/General
Books Division
605 Third Avenue,
New York, NY 10158-0012

In Canada:
22 Worcester Road
Rexdale, Ontario, M9W 1L1

ORDER NOW

JOHN WILEY & SONS
605 Third Avenue
New York, NY 10158-0012
Attn: M. Schustack

Please send me ____ copy(ies) of Krantz/OS/2: Features, Functions, and Applications (0 471-60709-6) $24.95 per copy plus applicable sales tax.

Please send me ____ copy(ies) of Conklin/OS/2: A Business Perspective (0 471-63503-0) $21.95 per copy plus applicable sales tax.

☐ Payment enclosed, Wiley pays postage/handling.
Bill my ☐ VISA ☐ MasterCard ☐ American Express

Acct. ____ Exp. ____

Signature ____

Name ____

Address ____

City/State/Zip ____

Price subject to change and higher in Canada.

CIRCLE NO. 140 ON READER SERVICE CARD

(continued from page 100)

parallel. You can also see pipeline architecture in the microprocessors of desktop computers. The Motorola 68030, for example, can fetch both instructions and data from on-chip caches, refresh the caches from off-chip memory, and ready the address for an off-chip fetch, all in parallel. Pipelining in CPU architecture and operating system design is a well-established technique. Extending this technique to a computer system as a whole presents a new paradigm, presently seen chiefly in supercomputers.

Both vector processing and array processing are single-instruction, multiple-data parallelism, and consequently they exhibit the traits of SIMD. SIMD approaches typically involve a high degree of parallelism; that is, many array or vector elements are processed at once. They restrict this parallelism to a low level, such as the level of the operation. They exhibit central control and strong synchronization. As a result of these traits, SIMD approaches concentrate most of the need for special algorithms at the system level and present no special problems in communication among the parallel elements.

MIMD Parallelism

In contrast, MIMD approaches usually involve fewer but more powerful processing elements, with a medium degree of parallelism and parallelism at a higher level—the level of the task. Control is distributed, and synchronization is only occasional. As a result, MIMD paradigms raise difficult problems in communication among the parallel tasks as well as higher-level problems in the design of algorithms.

In an MIMD architecture, different parts of an application program are given to different, independent, networked processors, each with its own instruction set, each capable of performing a sequence of instructions without supervision.

At the algorithmic level, MIMD parallelism means decomposing the problem into components that can be attacked in parallel via separate processes. This is a true paradigm shift; you envision the problem differently in MIMD parallelism from the way you envision it in SIMD parallelism or in a sequential paradigm. SIMD is a paradigm shift away from SISD, but MIMD is a shift away from SIMD and it's a bigger shift.

If Henry Ford's assembly line is

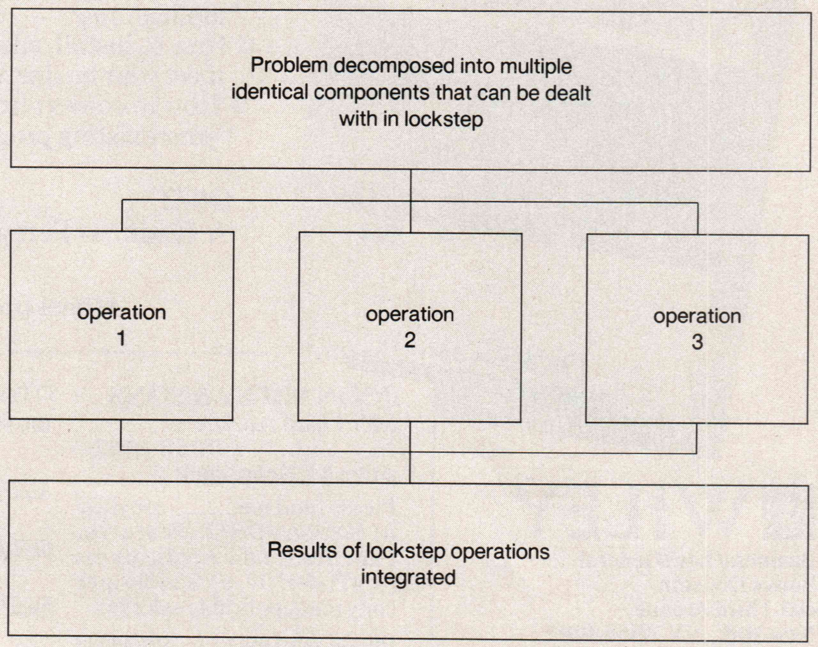


Figure 1: *Array processing applies the same operations to many data sets at the same time, achieving a lockstep parallelism.*

the metaphor for vector processing, then the array-processing metaphor is all those WACs plugging away at elementary arithmetic operations in lockstep back during World War II. MIMD parallelism is closer to the model of research within a scientific community, where, in pursuit of a common goal, individual researchers handle their own tasks, sharing information when it seems mutually beneficial—ARPAnet, including the networkers.

MIMD architectures (see Figure 3, below) are to date mostly experimental. Shared memory and nonshared memory architectures have been tried, as have been various topologies for the linking of the processors. The Denelcor Heterogeneous Array Processor and Cray X-MP are two commercial MIMD machines, and CalTech's Hypercube is a model that has been generating a lot of interest this year. But MIMD architectures are moving onto the desktop, and it is already possible to experiment with MIMD parallelism for the price of a fully loaded personal computer.

Foreseeing this possibility, Les Record, of Round Rock, Texas, sent me the following suggestion:

"Your discussion of problems in parallel computing brought to mind an idea for a major *DDJ* project: using minimal hardware, say 6 to 8 Z80s, interconnected in a simple network, present a series of articles using parallel techniques to solve simple (?!?) problems. If you don't want to present a hardware article (one every 2–3 years doesn't hurt), then perhaps some OEM might contract to build a minimal parallel system. Does this sound feasible? I think the hardware could be low-cost. The pay-

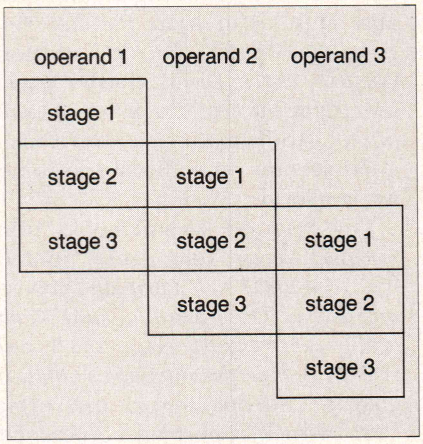


Figure 2: *Vector processing, or pipelining, achieves parallelism by overlapping the components of a multicomponent process, much as automobiles are built in parallel on a Detroit assembly line.*

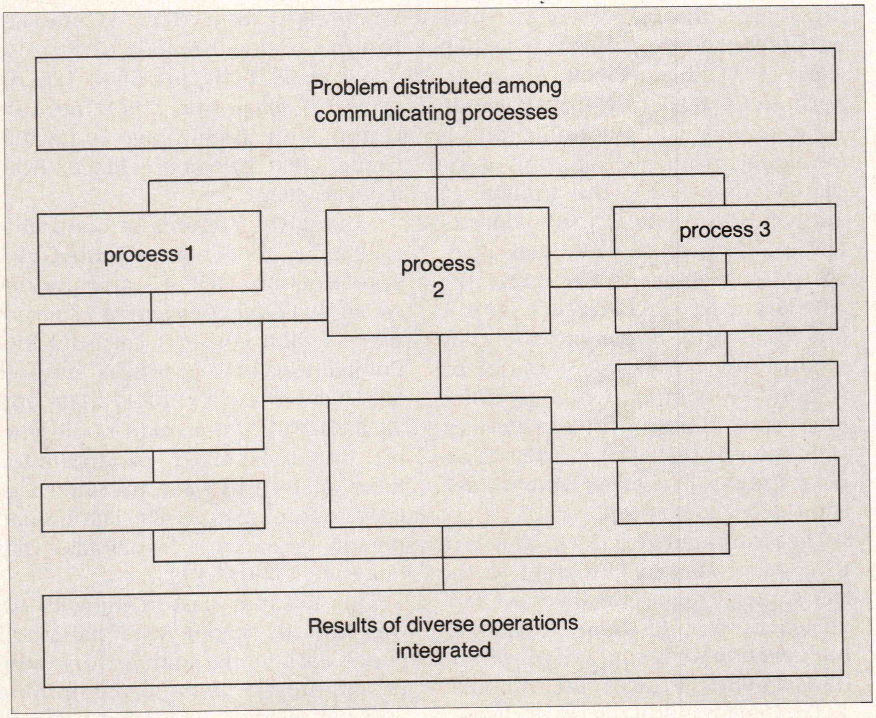


Figure 3: *MIMD (multiple instruction, multiple data) parallelism is typically asynchronous, with independent processes communicating with one another to achieve occasional synchronization.*

Sources

A column such as this can't begin to deal with its subject matter in the depth that, say, a column on C programming for MS-DOS can. I have only touched on many important issues in this installment. So here are some places where you can find the issues treated in greater depth.

You can find a good overview of parallel programming techniques and architectures in *Parallel Programming* by R.H. Perrott (Addison-Wesley, 1987). David Harel's excellent *Algorithmics: The Spirit of Computing* (Addison-Wesley, 1987) has a good section on some of the issues in parallel processing.

I also found several books from Springer-Verlag very useful, including *WOPPLOT 86, Parallel Processing: Logic, Organization, and Technology* (Springer-Verlag, 1987) and *PARLE, Parallel Architectures and Languages Europe* (Springer-Verlag, 1987), which report on conferences held, respectively, in Neubiberg, Federal Republic of Germany, in 1986 and in Eindhoven, the Netherlands in 1987.

The best source of information about the INMOS transputer is INMOS itself, through various technical notes. The U.S. contact is INMOS Corp., P.O. Box 16000, Colorado Springs, CO 80935; 303-630-4000.

Levco has video training and a developer group for learning about parallel processing, the transputer, and Levco's Macintosh boards. Levco is located at 6160 Lusk Blvd., Ste. C-100, San Diego, CA 92121; 619-457-2011. Levco's boards are not the only transputer implementation you might want to investigate; Definicon is another company that is doing interesting things with the devices. Definicon is located at 1100 Business Center Circle, Newbury Park, CA 91320; 805-499-0652.

Dick Pountain and David May give a very readable introduction to the transputer's high-level machine language, occam, in their *A Tutorial Introduction to Occam Programming* (INMOS/BSP Professional Books, 1987). Occam is still nearly undiscovered; this was one of only four books devoted to occam that I was able to find in a search of the entire University of California library system. Perrott's book also gives a clear introduction to Hoare's CSP and to occam as an implementation of CSP's central concepts as well as showing how parallel processing is implemented in a number of other languages, including Ada and Pascal Plus.

For explicit definitions of object-oriented programming, I drew upon "What Is Object-Oriented Programming," an invited lecture by Bjarne Stroustrup reprinted in *ECOOP '87: European Conference on Object-Oriented Programming* (Springer-Verlag, 1987); *Object-Oriented Programming: An Evolutionary Approach* by Brad Cox (Addison-Wesley, 1986); and the August 1981 and August 1986 issues of *Byte*.

But object-oriented programming is better defined by example in such sources as *Smalltalk-80, the Language and its Implementation* by Adele Goldberg and David Robson (Addison-Wesley, 1983); *The C++ Programming Language* by Bjarne Stroustrup (Addison-Wesley, 1986); and the documentation for Digitalk's Smalltalk-V and The Whitewater Group's Actor. Digitalk Inc. is located at 9841 Airport Blvd., Los Angeles, CA 90045; and The Whitewater Group is at Technology Innovation Center, 906 University Place, Evanston, IL 60201.

Among the sources of continuing education for object-oriented programmers are OOPSLA and the new *Journal of Object-Oriented Programming*, which you can get from SIGS Publications, 310 Madison Ave., Ste. 503, New York, NY 10017; 212- 972-7055. OOPSLA, the main conference on object-oriented programming, takes place this year in San Diego, Calif., September 25–29. The contact person is Barbara Noparstak, Digitalk Inc.; 213-645-1083.

This column had nothing to say this month about artificial-intelligence paradigms, such as functional programming, logic programming, and neural nets. The least I can do is to mention some good sources in these areas.

I am looking forward to my first issue of a journal devoted to LISP, called *LISP and Symbolic Computation: An International Journal*, edited by Richard Gabriel and Guy Steele. You can order it from Jan Zubkoff, *LASC*, Lucid Inc., 707 Laurel St., Menlo Park, CA 94025; 415-329-8400. Steele and Gabriel are, among other things, the authors of two important books on LISP—*Common LISP: The Language* by Guy L. Steele, Jr. (Digital Press, 1984) and *Performance and Evaluation of LISP Systems* by Richard P. Gabriel (MIT Press, 1985).

The Prolog language gets its name from the exaggerated notion that you are programming in logic when you use it. A new book removes some of the exaggeration by showing how to use and extend Prolog to do true logic programming; it's *Computing with Logic: Logic Programming with PROLOG* by David Maier and David S. Warren (Benjamin/Cummings, 1988).

A journal on neural network technology is available from *(TK)*. A conference on neural nets will also be held in San Diego, Calif., from July 24-27. It's sponsored by the IEEE and you can get more information from Nomi Feldman at 619-453-6222.

The big conference for the artificial-intelligence community is AAAI, held this year in St. Paul, Minn., from August 22-26. Call 415-328-3123.

Philosopher of science Thomas Kuhn made the term paradigm mean so much in his landmark book *The Structure of Scientific Revolutions* (University of Chicago Press, 1962). The disclaimer quoted at the beginning of this column is from the postscript in the second edition of this book, published in 1970. Robert Floyd's Turing Award lecture appears in *ACM Turing Award Lectures, The First Twenty Years: 1966–1985* (ACM Press, 1987)—a wonderful book. Most of the lectures in it not only deserve reading but also reward rereading. —M.S.

(continued from page 103)

off would be exposure to software—hands-on! I might even be able to help."

I especially like that last sentence. I'm passing Les' suggestion along to Tyler and the gang. But if the goal is exposure to the programming problems of MIMD parallelism rather than the architectural issues, wouldn't it make sense to buy the hardware off the rack? This is now possible with the INMOS transputer.

Transputers

Interest in the transputer is growing. Last summer, the Macintosh SIG of the Software Entrepreneur's Forum polled its members for topics for future meetings, and the hands-down winner was the INMOS transputer.

The transputer is a 32-bit RISC chip designed by INMOS Ltd. for parallel processing. It includes a processor, local memory, and four dedicated I/O ports. Several transputers in a simple network can be used to implement MIMD parallelism. The transputer can switch between parallel tasks in a microsecond.

Transputers can be linked in a network, with a great deal of flexibility in the topology of the network and in the physical location of the nodes: transputers on the same network need not occupy the same circuit board, the same system bus,

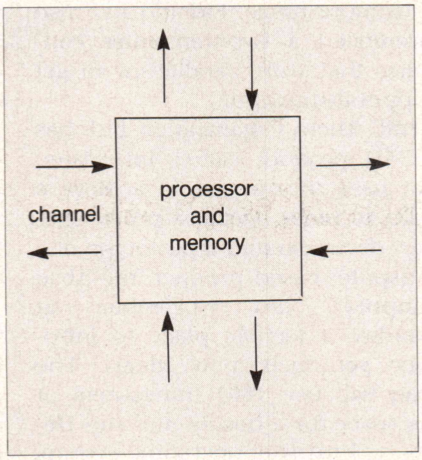


Figure 4: *One transputer processor. Independent processes can run in parallel on individual transputers, communicating with one another via the four channels each transputer has.*

(continued from page 105)

or the same city. Intertransputer links are point to point, so the size of the network is not limited by contention problems as it would be on a common bus. And the number of transputers in the network can be increased or decreased without altering the parallel program running on the network. (See Figure 4, page 105.)

The transputer is already being used in commercial products. Two companies have announced transputer-based boards that will significantly speed up laser printers.

The domain of MIMD algorithms is still wide open.

Both CSS Laboratories and Eidolon will be showing laser printer controller boards this year, each employing one T414 transputer as a coprocessor for throughput in the 8–20 page-per-minute range. Eidolon has also announced a two-transputer controller that uses parallelism to get 40 ppm throughput.

HRC Micro Organization Ltd. has a CAD product called Intervision that uses transputers to achieve a 10 to 40 times speed improvement. Atari demonstrated a prototype of a transputer-based product, the Abaq computer, last November (at Comdex, a terrible place to introduce genuinely new ideas). The Abaq will use T800 transputers (it has space for a dozen) and run Helios, a Unix-like operating system, with an MS-DOS emulation mode that should run DOS programs faster than an AT. Penguin Software is developing Unix-based transputer development tools. And Levco is selling transputer boards for personal computers.

It's Levco's boards I had in mind when I alluded to building a parallel-processing system from off-the-rack parts. Levco allows you to turn a Mac II or SE into a parallel-processing system with a package it calls TransLink. TransLink consists of a bus card, transputer modules, and either MPW-compatible development software (C compiler, transputer assembler, loader, linker) or the occam development system from INMOS. The transputer modules include 256K to 4 Mbytes of RAM in four SIMM sockets and one transputer (T414 15 MHz, T414 20 MHz, or T800 20 MHz with 64-bit IEEE floating-point support). The bus card for the Mac II can hold up to four transputer modules, whereas the SE card can hold only two.

Fully loaded with five TransLink Nubus-compatible boards, holding a total of 20 20-MHz T-800 transputer modules, each with from 256K to 4 Mbyte of its own RAM, a Mac II would reach a throughput of nearly 200 MIPS. (If only I weren't already in debt up to my neck for the Mac II. . . .)

Coming down from the clouds, I should explain that Levco's boards provide an opportunity for developers to explore the largely unknown territory of MIMD algorithms.

MIMD Programming

The domain of MIMD algorithms is still wide open. There are many hard problems to be solved, including questions of appropriate topology, general techniques for functional decomposition, and specific MIMD algorithms for familiar problems. Small-scale functional decomposition, with just a few processes running in parallel, can produce performance unattainable in any other way.

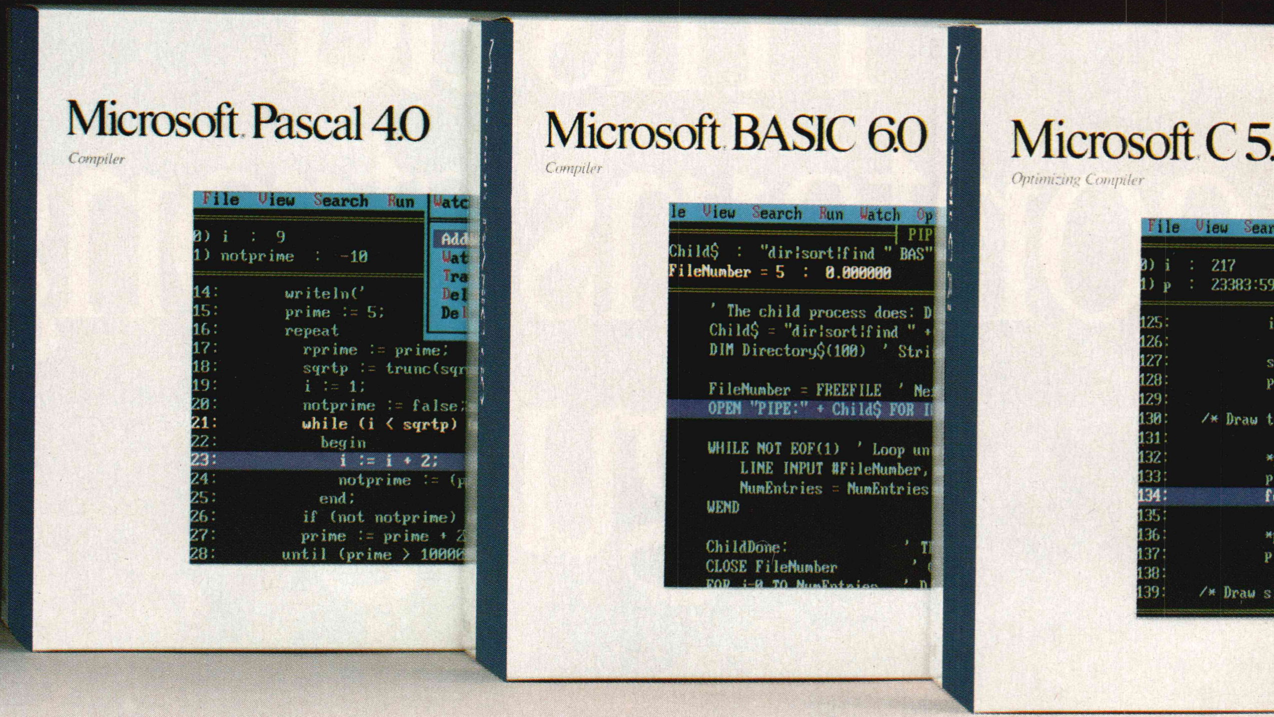
It is only fair to point out, though, that there are those who doubt the benefits of large-scale MIMD parallelism. Their arguments rest on a strict upper bound on those benefits: the limit set by Amdahl's law.

Amdahl's law states that parallel speedup (the ratio of the speed of processing using any parallel technique whatsoever to the speed of the strictly sequential approach) is bounded above by $100/[(100-f)+f/p]$,

WARNING:

This ad contains strong language.

Introducing Mic
for OS/2



The people who co-developed the industry's most powerful personal computer operating system are now proud to announce programming languages to match.

Introducing Microsoft® Macro Assembler 5.1, C 5.1, Pascal 4.0, FORTRAN 4.1 and BASIC Compiler 6.0.

Five industrial-strength, stand-alone languages that combine the implementation flexibility you've enjoyed under MS-DOS® (which, of course, they still support) with the advanced capabilities you've anticipated from OS/2.

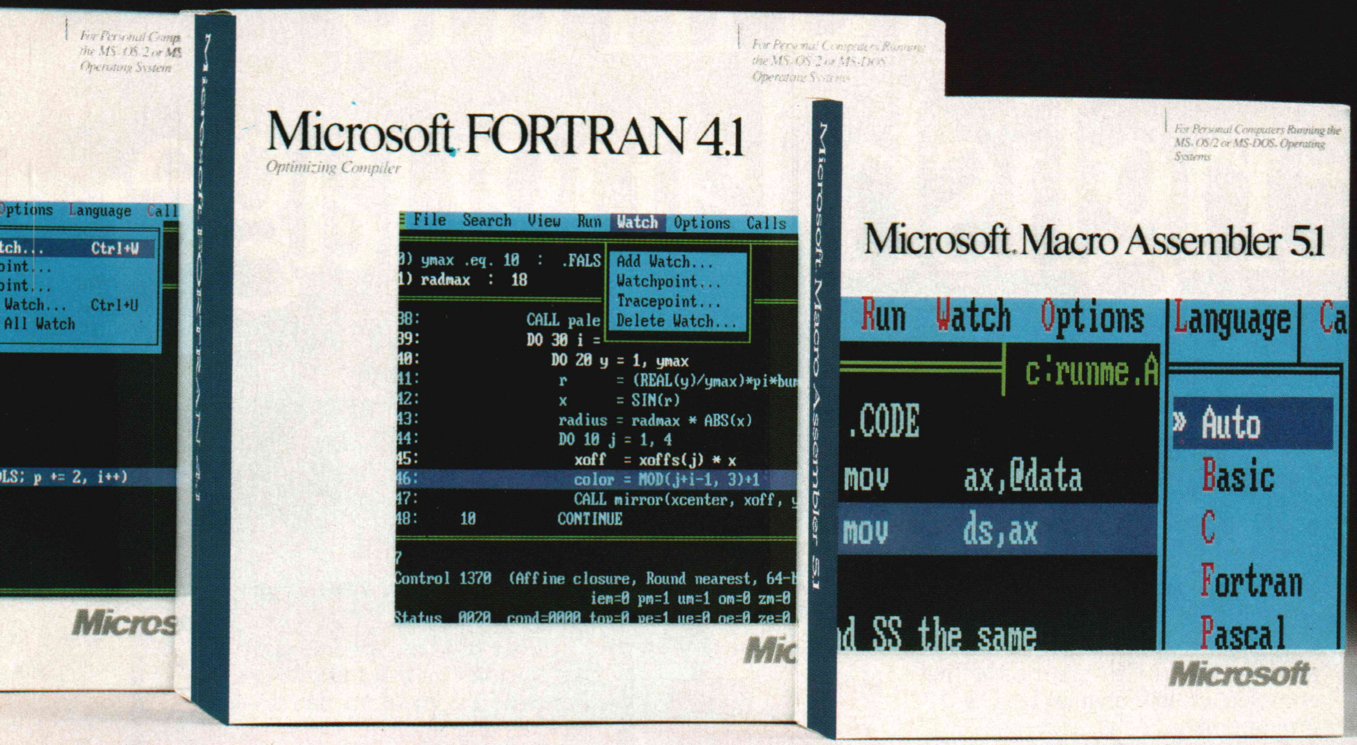
Capabilities such as the ability to develop large, sophisticated applications which go beyond the 640K barrier, taking advantage of up to 16MB of RAM, and utilizing the potential of today's microprocessors.

Just like their MS-DOS predecessors, these five new languages are equipped with powerful, professional features you work with, not around:

Support of direct calls to the operating system, and inter-language calling for mixing multiple languages on the same project.

Access to OS/2 system calls and a full complement of utilities, including an incredibly fast incremental linker and the

rosoft Languages systems.



first protected mode programmer's editor that works equally well in real mode.

Microsoft CodeView,® our popular, advanced debugger that lets you untangle program logic at the source code level, no matter what code you're using.

(It even lets you debug protected mode programs up to 128MB of virtual memory, and larger programs than ever before in real mode.)

As the perfect complement to our new languages, we're also offering the Microsoft OS/2 Programmer's Toolkit.

It contains a parameter-by-parameter breakdown of all OS/2 system calls and samples to get you started.

All the tools you need for turning out larger, more powerful, more complex OS/2 applications.

(And, incidentally, all the tools we rely on for creating our own commercial software.)

For the name of your nearest Microsoft professional languages dealer, simply call 800-541-1261, Dept. B99.

Ask him for some more information on our OS/2 family.

He'll show you some languages you can really swear by.

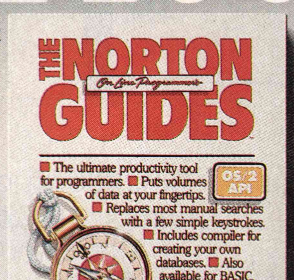
And now for the strongest word in the English language:

Free.

The Norton On-Line Programmer's Guide for OS/2 API is the first complete on-line manual for OS/2 programming.

Instead of thumbing through pages of documentation, it's all there at your fingertips with a few simple keystrokes.

Normally it costs $150, but it's yours free when you acquire the Microsoft OS/2 Programmer's Toolkit and one of the high level languages listed opposite (an upgrade is fine).



Please send me my free copy of the Norton Guide for OS/2 API. I enclose a copy of my dated sales receipts and my registration cards.*

The high level language I have licensed is (please check):

- ☐ Microsoft C Optimizing Compiler 5.10.
- ☐ Microsoft FORTRAN Optimizing Compiler 4.10.
- ☐ Microsoft Macro Assembler 5.10.
- ☐ Microsoft Pascal Compiler 4.00.
- ☐ Microsoft BASIC Compiler 6.00.

Redeem to: Norton Guide for OS/2 API offer, Microsoft Corporation, 13221 SE 26th, Suite L, Bellevue, WA 98005.

Name: ____________________

Address: ____________________

City: ____________ State: ____________ Zip: ____________

Daytime telephone: (   ) ____________________

If you have any questions about this offer, call (800) 426-9400. In WA, (206) 882-8088. B99

Microsoft

*Registration cards are not required for upgrades. This offer is only valid in the 50 United States. It is not valid with any other offers, and is effective only for purchases from 4/1/88 through 6/30/88. The coupon must be redeemed by 7/31/88. Please allow 4-6 weeks for delivery.

(continued from page 106)

where f is the percent of the total work that can be done in parallel and p is the number of processors. The value of f cannot be 100 because some communication and control overhead is necessarily not parallelizable. This means that, with five processors and an algorithm that is 75 percent parallelizable, the maximum speedup attainable is only 2.5. Adding more processors will only increase the speedup gradually toward a final limit (for this particular algorithm) of 4.0. Given that communication overhead can increase rapidly with the number of processors in an MIMD system, the value of additional processors drops quickly to nothing in this scenario.

Nevertheless, even large-scale MIMD parallelism is worth looking into. The problem with the scenario just discussed was the 75 percent parallelizability. As the value of f in Amdahl's equation approaches 100, the entire expression approaches linearity; that is, n processors yield an n-fold speedup. There are experimental results showing that in interesting cases near linearity is in fact attainable. For one example, see McBurney's comments in *PARLE, Parallel Architectures and Languages Europe* (Springer-Verlag, 1987).

Relative speedup, which is the speedup due to parallelization minus overhead, approaches linearity as the number of processes (that is, tasks, not processors) increases (but it has to increase a lot). Of course, it is exactly the nonparallelizable overhead that sets the limit in Amdahl's equation; but when, as is the case in a wide variety of problems discussed by McBurney, the overhead cost becomes negligible with increasing problem size, then even large-scale MIMD parallelism suddenly makes sense.

MIMD puts more burden on the application programmer to partition the program into appropriate components for parallel execution. Languages have been developed to facilitate this partitioning, some of them deriving from a model developed by C.A.R. Hoare called Communicating Sequential Processes, or CSP.

CSP

Hoare's model is of two or more independent processes, each consisting of a sequence of instructions executed sequentially. The processes are no different from programs written in any conventional sequential programming language, except when they must communicate with one another. In Hoare's model, this can only happen when each of two processes desires to communicate with the other process, and he calls this synchronization of desires a rendezvous.

Hoare's rendezvous is a strict synchronization, and by itself it would defeat one of the canons of parallelism, which is to keep all processes busy as much of the time as is possible. To free a process that might otherwise be locked up waiting for a rendezvous, Hoare adopted nondeterministic alternation and repetition constructs from Dijkstra.

These nondeterministic constructs require something called a guard, which is a Boolean expression that precedes a command. Only if the guard is true is the command executed, but its truth does not guarantee execution. The guarded alternation construct causes at most one of a set of guarded commands to be

(continued from page 111)

executed, with that one selected at random from those commands with true guards. The guarded repetitive construct causes all commands with true guards to be executed until no guards are true.

Guarded alternation and repetition allow a process, for example, to pass on data to any other available process or to get data for processing any time any other process has data to deliver.

Together, the concepts of sequential processes communicating via rendezvous and nondeterministic alternation and repetition make up a model for MIMD parallelism that is both provable and a solid basis for a programming language. One such language, and the natural one for developing transputer-based software, is occam.

Occam

"Occam's Razor slices things down to simplest causes. Single causes have a fair chance of being right."—from "Occam's Scalpel" by Theodore Sturgeon (*Worlds of IF*, August 1971).

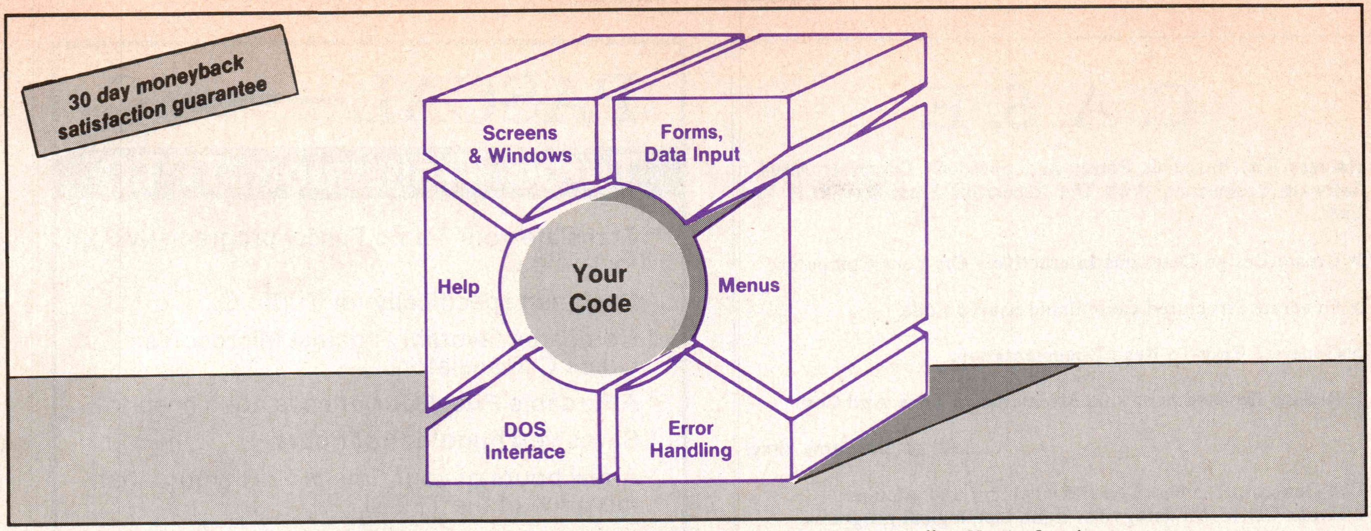
Occam's razor (or sometimes Ockham's razor) is the principle of ontological economy, attributed to William of Ockham (or Occam), circa 1285–1349, an English Franciscan, heretic, and philosopher best known among philosophers for his antirealist interpretations of universals. To the rest of us, if he is known at all, it is for Occam's razor, which states that, in explaining nature, entities should not be multiplied beyond necessity: the simplest explanation is the best.

The programming language occam (with a lowercase *o*) was designed by people at INMOS for writing parallel programs to run on the INMOS transputer processors. Occam supports parallel processing with only a few new programming entities. So clean is the occam implementation that major portions of the occam development system have been formally proved correct.

Occam implements the key concepts of CSP directly by the use of constructors. Sequences of commands (also called primitive processes) are grouped into sequential processes using the constructor *seq*, concurrency among such processes is indicated by the constructor *par*, and nondeterminism is introduced with the constructor *alt*. Nondeterministic alternation and repetition are implemented using the *alt* constructor in conjunction with Pascal-like alternation and repetition constructs such as *if* and *while*. Communication between processes follows the CSP rendezvous model, with two processes choosing a common channel on which to signal or pass data.

Although occam is a high-level language, it is also in some sense the machine language of the transputer, which was designed using occam. The two are closely linked. In particular, occam processes are run on individual transputers, and occam channels map directly onto the physical transputer links.

In Example 1, page 115, I give a taste of occam code. The three processes INPROC, BUFFPROC, and OUTPROC run in parallel, with INPROC gathering data and passing it to BUFFPROC, which in turn passes it to OUTPROC for output. Communication between processes in CSP occurs at a rendezvous and there are two in this example: when BUFFPROC calls INPROC and vice



C-Worthy Interface Library helps you smoothly pull together all aspects of an excellent Human Interface.

C Programmers: Wrap an Exciting, Bullet-Proof Interface Around Your Code Quickly.

Introducing... C-Worthy® Interface Library

The only human interface package you need. That's what our customers are telling us. One early adopter, Novell, Inc. uses it exclusively in the development of their NetWare® Utilities, which reach over 500,000 users. You see, C-Worthy Interface Library is the only library available to handle every aspect of your program's human interface, all in one package. Now your programs will have a consistent look and feel. You no longer have to integrate pieces of libraries from different manufacturers.

As important as you know users are, you often don't have the time to heavily invest in writing routine code. And that's OK, because with over 400 tight, ready-to-use functions, C-Worthy Interface Library takes care of the tedium and lets you spend your time doing what you enjoy. Concentrate on the heart of your application — features that make it unique, special. Let C-Worthy Interface Library do your:

- Menus
- Error Handling
- DOS Interface
- Context Sensitive Help
- Screens, Windows
- Forms, Data Input (optional)

You control color, size, border, location, etc. And if there's anything you want to change, you can. Source is available to provide you with the flexibility you need. And you can distribute your applications freely, with no royalties.

C-Worthy Interface Library requires hard disk media with 256K RAM. MSDOS 2.0 + and IBM PC, or compatible, TI Professional, NEC APC III, or VICTOR 9000. C-Worthy is a registered trademark of Custom Design Systems, Inc.

Tech Specs

- Compilers: Microsoft 3.0+, Quick, Turbo, Lattice. All models.
- 350+ functions written in C, 75+ in Assembler.
- Menus: Fully support pop-up, Lotus style, MS Windows style (pull-down), pull-up.
- Errors: DOS, program, and user.
- DOS Interface: 62 functions. File handling, dir. and drive management, date & time conversion, wildcards, more.
- Help: System and context sensitive.
- Screens: Screen display, color palettes, save, restore, scroll, more.
- Windows: Exploding, tiled, pop-up, overlapping. Direct video access and virtual. Up to 50 active at any time.
- Keyboard Handling: Regular, function, interrupt, background procedures.
- Editing: String and word wrap text.
- Form Interface Library: 118 functions. Over 15 field types, and user definable field types. 3 levels of data validation: type, multiple field ranges, optional validation procedures. Hide, lock, or secure a field. Optimal field movement.
- Foreign Languages: All text messages in separate files for easy translation.
- Compatible with MS Windows.
- OS/2 special overlay when released.
- Machines: Autodetect for MDA, CGA, EGA, VGA, TI, AT&T, Victor.
- No royalties.

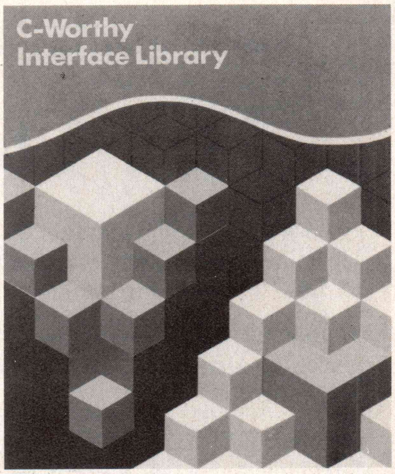
"I heartily recommend this package."
— David A. Schmitt, president, Lattice, Inc.
Over 400 developers in 16 countries already use it.

Thorough Documentation

Indexed alphabetically and by category, the 700+ page Reference Guide includes for each function: an example, description, calling conventions, return values, and related functions. The 250 page User's Guide gets you going with its tutorial and "Getting Started" sections.

CIRCLE NO. 229 ON READER SERVICE CARD

"C-Worthy is a comprehensive C library whose time has come. I heartily recommend it as your next purchase." —Computer Language, 8/87



C-Worthy Interface Library:

Object only	**$ 195**
Form Interface Library add-on	**$ 100**
Object with Forms	**$ 295**
Object with Forms & Library Source	**$ 495**

Please specify compiler and version when ordering.

To Order Call
(800) 821- 2492
in MA (617) 337-6963

Solution Systems™

541-D Main Street, Suite 410
South Weymouth, MA 02190

(continued from page 112)

versa, and when BUFFPROC calls OUTPROC and vice versa. Note that indentation is not optional in occam.

I hope this taste of parallel paradigms has been useful and entertaining. Thinking through parallel approaches can be enlightening even if you never expect to work on a parallel system. Several programmers have pointed out the possibilities for learning new sequential approaches from studying parallel techniques. And it adds to your paradigmatic breadth.

WHOOPS, or What is Object-Oriented Programming?

One programming paradigm that has become extremely popular of late is object-oriented programming. *Object-oriented* is the current vogue word, displacing *structured* as synonymous with whatever is good and true and beautiful in programming. Smalltalk, Actor, Simula67, and C++ are described as object-oriented; there have been claims that Ada, LOOPS, and APL are object-oriented languages; and there is much talk of object-oriented design in Pascal, Modula-2, C, and Forth. Just what is the object-oriented paradigm? What features does a language need to have in order to be said to support the paradigm? And what does it mean to say that one is programming in the paradigm?

Bjarne Stroustrup has come up with a set of definitions that help to clarify the issue, at least if you accept them. I do, chiefly because they give a clear picture of object-oriented programming as a paradigm, indicating what kinds of puzzles the universe sets an object-oriented programmer.

Stroustrup takes pains to distinguish the object-oriented paradigm from the paradigms of data hiding and data abstraction. Data hiding, he says, boils down to using modules. You can achieve the effect of data hiding in C, but Modula-2 makes the module a fundamental language construct. Stroustrup says that Modula-2 supports data hiding but C only enables it.

Data abstraction according to Stroustrup means programming with user-defined types. Any programming language that provides the means for doing this supports the data abstraction paradigm—Ada and C++, for two examples.

One thing that the data-abstraction paradigm does not permit is expressing a distinction between the properties of a type and the properties of instances of the type. Object-oriented languages, Stroustrup says, are those that support expressing this distinction, such as Smalltalk. The mechanism for doing this is inheritance.

Object-oriented programming is, for Stroustrup, just programming using inheritance, and it is, roughly, a superset of the other paradigms. Smalltalk is certainly an object-oriented language, or rather an object-oriented environment. (Strictly speaking, no language can be object-oriented by itself; object-oriented programming requires support from a programming environment as well as support from a language.) I'm not sure where this leaves HyperTalk, which seems to have an inheritance structure but which does not allow creation of new types of objects. In any case, he breaks down the object-oriented paradigm further, as follows:

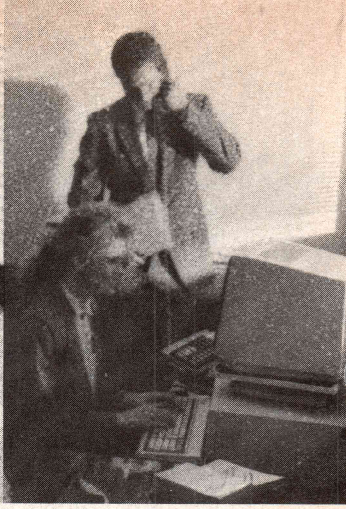
- decide which classes you want
- provide a full set of operations for each class
- make commonality explicit by using inheritance

The benefits of the object-oriented paradigm come from the exploitation of the commonality among types, and identifying commonality

```
INPROC::

        ...
        (* input X*)
BUFFPROC ! X

        ...

OUTPROC::

        ...

        BUFFPROC ? X
        (* output X *)
```

Example 1: *Communicating processes in CSP.*





One Language For V

BBX® Specifications:

Ease-of-use—BB^x is the fastest, most powerful development tool available for business oriented program creation. Programmers can write code in minutes.

Execution time—BB^x's partially compiled format provides enhanced execution speed.

Easy Maintenance—BB^x is an interactive programming language, with a trace facility and a full screen editor which makes program maintenance a snap.

Portability—BB^x runs under UNIX and other operating systems without recompilation.

Compatibility—BB^x is an enhancement of the Business BASIC language, an industry standard, giving you access to thousands of applications.

Supportability—Program maintenance utilities and complete documentation save considerable time and money. It lets you build and support applications easily.

Utilities—A complete set of BB^x utilities are provided for program and file management.

Conversion—A complete set of conversion utilities are provided with every BB^x package.

Features

Math Functions
- 14 place precision and computational accuracy
- Floating point conversion
- Task specified rounding precision
- Binary to decimal conversion
- Long function names
- Dynamic arrays

String Functions
- Numeric to string conversion
- String manipulation
- No string length restriction

I/O Functions
- Windowing
- I/O mnemonics
- Device independent verbs
- X,Y cursor addressing
- Masking
- Soft key loads
- No record length restrictions
- BB^x file sizes are limited only to the size of the available media

File Structures
- INDEX
- KEYED
- MKEYED
- SERIAL
- SORT
- PROGRAM
- STRING

System Structure
- Multi-tasking - which provides record and file level locking
- Program overlay
- Public programming which provides:
  - Local variables
  - Dynamically called sub-programs
  - Argument passing
  - Automatic public program drop from memory at exit
  - Public program in memory lock option

Language Structure
- Interactive program development
- Online syntax checking
- Compound statements
- User defined functions
- Unlimited nesting
- Remote I/O lists
- Program self modification
- Case insensitive console mode
- Various debugging tools

BB^x Utility Set
- File Browse
- Create Data Bundle
- Calculator
- Clear Workspace
- Program Compare
- Copy File
- Define/Redefine File
- Directory Listing
- Erase File
- Generate Filelist
- Program List/Cross Reference
- Move File
- Program Renumbered
- Rename File
- File Resizer
- Execute O/S Shell Command
- Search and Replace Program
- Color & FUNC Key Setup
- Time/Date Examine/Set
- Utility Menu
- Visual Utility Interface
- BXSND/BXRCV conversion utilities

Its portability crosses all operating environments, and now its performance is crossing all oceans.

Around the world, the industry's best and brightest programmers are discovering the astonishing power that BB^x brings to Business BASIC. Write your program once, and have complete movement to MS/PC-DOS, OS/2, UNIX/XENIX, AIX, IX370 and VMS.

This year, over 50,000 copies of BB^x are performing throughout the United States, Canada, Europe, Asia and South America.

Commitment to innovation, development within industry standards and technological leadership have grown BB^x around the globe.

In 1988, aggressive marketing and uncompromising customer support will continue to compliment our success, and expand the BB^x standard among many of the world's most respected companies.

Get in touch with one of our world distributors, and feel the pulse of the power of BB^x!

Die Portabilität schlägt sämtliche, bisher bekannte und unbekannte, EDV-Emgebungen. Die Leistung überzeugt inzwischen die gesamte EDV-Industrie.

Weltweit entdecken die besten Software-Entwickler die erstaunliche Leistung von BB^x, mit der Business BASIC bereichert wird. Die Anwendungen werden nur einmal entwickelt und laufen ohne Änderungen oder Anpassungen auf MS/PC-DOS, OS/2, UNIX/XENIX, AIX, IX370 oder VMS.

Mehr als 50.000 BB^x-Lizenzen stellen die Leistung in den USA, Canada, Europa, Asien und Süd-Amerika unter Beweis.

Zur Innovation nach Industrie Standard Spezifikationen verpflichtet, und mit dem Ziel nach technologischer Führung, wächst BB^x um die Welt.

Mit aggressivem Marketing ohne Kompromisse im Bereich Kundenservice, wird der Erfolg von BB^x in 1988 fortgesetzt. Es steht auf sämtlichen Systemen namhafter Computerhersteller zur Verfügung und stellt seine Akzeptanz bei den anspruchvollsten Anwendern unter Beweis.

Kontaktieren Sie unsere Vertretungen in aller Welt. Entdecken Sie die Schlagkraft von BB^x!

BB^x PROGRESSION/2® is available for Intel Based Computers, Altos, Arete, AT&T, PCS Cadmus, Computer Consoles, Convergent Technologies, Counterpoint/MultiTech, Cubix, Data General, Digital Equipment, Fortune, Honeywell, Hewlett Packard, ICL, Motorola, Nixdorf, Prime, Pyramid, Rexon, Sanyo, Sequent, Siemens, Texas Instruments, Unisys, and the IBM family of products. BASIS is continually adding new systems.

orld Class Business.

Portable, il franchit tous les cadres d'opération, et sa performance traverse, maintenant, tous les océans.

Dans le monde, les meilleurs et les plus brillants programmeurs de l'industrie découvrent l'étonnante puissance que BB^{x} amène au BASIC des affaires. Ecrivez votre programme une seule fois, et accédez totalement à MS/PC-DOS, OS/2, UNIX/XENIX, AIX, IX370 et à VMS.

Cette année, plus de 50 000 copies de BB^{x} fonctionnent aux Etats-Unis, au Canada, en Europe, en Asie et en Amérique du Sud.

Un esprit constant d'innovation, un développement conforme aux normes de l'industrie, et une position de leader dans le domaine technologique, tels sont les atouts qui ont contribué à la croissance de BB^{x} dans le monde entier.

En 1988, un marketing dynamique et un appui inconditionnel à notre clientèle continueront à couronner notre réussite, et à étendre le standard BB^{x} à de nombreuses sociétés parmi les plus respectées au monde.

Contactez l'un de nos distributeurs mondiaux et découvrez la puissance de BB^{x}!

Su portabilidad traspasa todos los medios de operación y ahora su funcionamiento esta cruzando todos los océanos.

Los mejores y más brillantes programadores del mundo, están descubriendo la asombrosa potencia que BB^{x} ofrece al negocio BASIC. Escriba su programa una vez y tenga movimiento completo a MS/PC-DOS, OS/2, UNIX/XENIX, AIX, IX370 y VMS.

Este año, más de 50,000 copias de BB^{x} estan funcionando en Estados Unidos, Canadá, Europa, Asia y América del Sur.

Empeño de inovación, desarrollo en los estandards de la industria y superioridad tecnológica han hecho crecer a BB^{x} en todo el mundo.

En 1988, mercadotecnia agresiva y apoyo constante a nuestros clientes seguirán complementando el exito y desarrollo de BB^{x} entre las compañías más respetadas del mundo.

Comuniquese con uno de nuestros distribuidores mundiales y sienta la potencia de BB^{x}!

World Distributors:

Edias Hans Kirchhoff & Co. KG
Pfingstbornstr. 25, 6200 Wiesbaden
TEL: (06122) 2016
FAX: (06122) 16505
TLX: 418-2563 edia d

West Germany, The Netherlands
Austria, Switzerland
Denmark, Luxembourg, Belgium
England, Italy

Multisys
Torgeir Vraas Plass 5A
3044 Drammen Norway
TEL: (03) 83.86.05
FAX: (03) 89.02.53

Norway, Sweden, Finland,
Greenland, Iceland

J. P. Brown and Associates
780 Gordon Baker Road
Willowdale, Ontario M2H 3B4
TEL: (416) 494-0472

Canada

PI Informatique
8, rue Benjamin Constant
75019 Paris, France
TEL: (01) 40.05.10.65
TLX: 214.583
FAX: (01) 40.05.99.63

France, Spain, Portugal

Softech Pty. Limited
10 Eileen Road, Blairgowrie
Randburg 2194, South Africa
TEL: (011) 787-8839
FAX: (011) 886-3890
TLX: 422669 S.A.

South Africa

Risegold Pty. Ltd.
678 Paramatta Road
Croydon, N.S.W. 2132
Australia
TEL: (02) 799-6622
FAX: (02) 799-9090

Risegold Pty. Ltd.
86 Havelock Street
West Perth
Western Australia 6005
TEL: (09) 481-0607
FAX: (09) 481-3162

Australia

Tempo Computadoras
Av. Americas #670
Guadalajara 44680
Jalisco, Mexico
TELS: 30-28-45
30-28-46
30-28-86

Infotel, S.A., de, C.V.
Laguna de Mayran
No. 258 3 piso
Col. Anahuac C.P. 11320
Mexico, D.F.
TELS: 05 45 6730 al
05 45 6734

Mexico

In the United States:
BASIS Incorporated
P.O. Box 20400
Albuquerque, New Mexico 87154
TEL: (505) 821-4407
FAX: (505) 821-1625

BASIS™
INCORPORATED

10% DISCOUNT!
ANY ORDER BY A NEW CUSTOMER, ACCOMPANIED BY THIS ADVERTISEMENT, QUALIFIES FOR A 10% DISCOUNT!
Call TOLL FREE directly to our Order Department,
1-800-423-1394

DDM

BB^{x} PROGRESSION/2®, BB^{x} and BASIS Incorporated are trademarks and/or service marks of BASIS Incorporated, Albuquerque, New Mexico. All references to computer systems and software products contained within this advertisement recognize the trade and/or service marks of the corresponding manufacturer and holder of the trade and/or service mark.

CIRCLE NO. 90 ON READER SERVICE CARD

PROGRAMMING PARADIGMS

(continued from page 115)

in the problem is the chief task that the paradigm sets the programmer.

Stroustrup's definition of object-oriented programming is not the same as David Robson's in Robson's classic *Byte* article of August 1981, in which he presents the class/instance distinction (and consequently inheritance) as optional. But Robson seems to be defining a concept rather than a paradigm. It also does not match Brad Cox's view of the centrality of the software IC metaphor to object-oriented programming. And it is at variance with Geoffrey Pascoe's view, expressed in *Byte* in August 1986 that information hiding, data abstraction, dynamic binding, and inheritance are all defining features of object-oriented programming. But Stroustrup argues that information hiding and data abstraction are subsumed within inheritance.

One thing that Stroustrup's definition does provide is a picture of the kind of puzzles the object-oriented programmer faces as a result of being an object-oriented programmer. The object-oriented programmer selects a set of classes, provides operations for each class, and then sets out to identifying commonality in the problem in order to make it explicit by using inheritance.

It's explicitly a definition of object-oriented programming as a paradigm.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call 415-366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

A DEAL YOU CAN'T REFUSE

FREE* Turbo C® (Borland) or **FREE* QuickC™** (Microsoft) or **Microsoft® C $100 REFUND** on your present MS C compiler with purchase of C Starter or C Business Library

* If you ALREADY own one, get a **FREE REFUND**... See special offer. (Limited time offer)

C *WHY* YOU CAN'T REFUSE

- **A C COMPILER** *without a good add-on library is like a PC without a keyboard... it won't do what you want it to do.*
- **GAIN C POWER** *Add capabilities your compiler library does NOT have. e.g.:*
  - *New! Qwick Menuing—full 1-2-3 like menus & more*
  - *Flexible powerful windowing + new Qwick windows*
  - *Powerful cursor, video and attribute control*
  - *Time and date arithmetic*
  - *Sample code and working examples*
  - *New! Qwick Data Entry with dialog boxes*
  - *Formatted, fully validated data entry*
  - *Display default field values*
  - *Calculator style entry option*
  - *500 functions you need*
- **SAVE TIME, TIME, TIME:** *man-years on development, calendar months on schedule!*
- **SAVE MONEY:** *Lowest Cost, Highest Quality Library/Windows Available!*
- **SMALLER PROGRAM SIZE:** *your application program can be up to 50% smaller!*
- **EASY** *for beginners!* **POWERFUL** *for professionals!*
- **INSTANT INSTALLATION UTILITY** *included!*
- **SUPERB DOCUMENTATION:** *time saving, helpful, clear, complete, instructive.*
- **BUSINESS USERS: FREE** *3 machine site license (C Library & Power Windows).*
- **FULL SOURCE CODE! NO ROYALTIES** *on products you develop.*
- **FREE UTILITY:** *To convert Turbo Pascal code to C code.*

BORN AGAIN DISCOUNT
If you're not happy with your present tools for C, or would like to upgrade to Entelekon, ask about the **Big** Born Again Discount.

SAVE MONEY! SAVE TIME! DON'T WAIT! ORDER NOW!

SATISFACTION GUARANTEED **(Direct from Entelekon only)** **CALL (713) 468-4412**

POWER WINDOWS™

MOST POWERFUL YET
POP-UP/PULL DOWN/OVERLAP
Menus/Overlays
Help Screens
Messages/Alarms
ZAP ON/OFF SCREEN
FILE-WINDOW MANAGEMENT
Horizontal & Vertical Scrolling
Word Wrap & Line Insertion
Cursor/Attributes/Borders

Many types of menus. Highlighting. Move data between files, keyboard, program and windows. Status lines. Change size/location/overlapping. Move/add/delete/cascade windows.

6 diskettes $159.95

C FUNCTION LIBRARY

BEST YOU CAN GET
HUNDREDS OF FUNCTIONS
FULLY TESTED
BETTER FUNCTIONS

Most complete screen handling plus graphics; cursor/keyboard/data entry, 72 string functions with word wrap; status and control; utility/DOS BIOS/time/data functions; printer control & more. Special Functions.

14 diskettes $159.95

C BUSINESS LIBRARY

INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM

ALL for $299.95
(A $500.00 VALUE)

B-TREE LIBRARY & ISAM DRIVER

POWERFUL DATA MANAGER
FAST! EASTY TO USE!
16.7 MILLION RECORDS/FILE
16.7 MILLION KEYS/FILE
Fixed/Variable length records.

Fast B-tree indices. Add/remove keys Find first/last/next/any key. Find keys by Boolean selection. Read/write/delete or add records to file.

Full source. No royalties $129.95
Multi-User option available.

C STARTER PACKAGE

INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C (20 DISKETTES)

ALL for $199.95
(A $370.00 VALUE)

* SPECIAL OFFER

Free Turbo C or QuickC with purchase of C Starter Package, C Business Library, C Function Library or Power Windows. Even if you already own Turbo C or QuickC, we will refund up to the full purchase price of one of these packages with the purchase of C Starter Package or C Business Library.

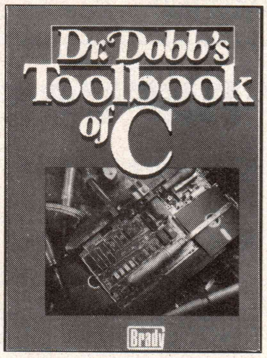
Entelekon™
SINCE 1982

12118 Kimberlev. Houston, TX 77024 713-468-4412 VISA-MASTERCARD-CHECK-COD

CIRCLE NO. 124 ON READER SERVICE CARD

Don't Get Lost in the Troughs Cruise the High C's

Dr. Dobb's Toolbook of C



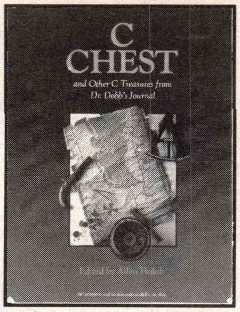
by the Editors of
Dr. Dobb's Journal
of Software Tools

This authoritative reference contains more than 700 pages of the best C articles and source code from Dr. Dobb's Journal of Software Tools, along with new material by C experts. The level is sophisticated and pragmatic: appropriate for professional C programmers. You'll find hundreds of pages of useful C source code, including a complete compiler, an assembler, and text- processing utilities. Highlights include:

- James E. Hendrix's famous Small-C Compiler and New Library for Small C
- Also, James E. Hendrix's Small Mac: An Assembler for Small C and Small Tools: Programs for Text Processing
- All of Anthony Skjellum's C Programmer's Notebook columns distilled into one thought-provoking chapter

Book **Item #615-3** **$29.95**

C Chest and Other C Treasures



by Allen Holub

This comprehensive anthology contains the popular "C Chest" columns from *Dr. Dobb's Journal of Software Tools,* along with the lively philosophical and practical discussions they inspired, plus other information-packed articles by C experts.

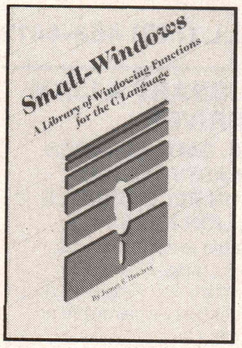
Topics covered include: pipes, wild-card expansion, and quoted arguments; sorting routines; command-line processing; queues and bit maps; utilities such as ls, make, and more; expression parsing; hyphenation; IBM cursor control and an Fget that edits; redirection; accessing IBM video display memory; trees; an AVL tree database package; directory traversal; sets; shrinking .EXE file images; hashing, expressions, and roman numerals; and statistical applications of digital low-pass filters.

Other treasures include: a variable metric minimizer; Fgrep; a peephole optimizer; and curve fitting with cubic splines.

All subroutines and programs are written in C, and are available on disk with full source code. MS-DOS format.

Book & Disk (MS-DOS)	**Item #49-6**	**$38.95**
Book	**Item #40-2**	**$23.95**

Small Windows: A Library of Windowing Functions for the C Language



by James E. Hendrix

Small-Windows is a complete windowing library for C (Microsoft 4.0/5.0, Turbo C 1.5, Lattice C 3.1, and Small-C). The package includes:

- 18 video functions written in assembly language
- 7 menu functions that support both static and pop-up menus
- 41 window functions to clean, frame, move, hide, show, scroll, push and pop windows

Complete Source Code Included — Microsoft 4.0/5.0, Turbo C 1.5, Lattice 3.1 and Small C are all supported

A file directory facility illustrates the use of window menu functions and provides file selection, renaming, and deletion capability. Two test programs are provided as examples to show you how to use the library and the window, menu, and directory functions.

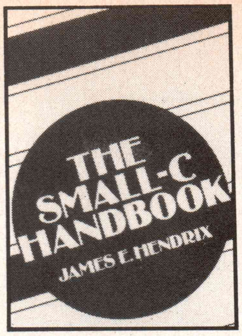
The **Small-Windows** package is available for MS-DOS systems, and Microsoft C Version 4.0/5.0, Turbo C1.5, Lattice C 3.1and Small-C compilers. Documentation and full C source code is included.

Manual & Disk (MS-DOS)
(Microsoft C, Small-C, Lattice C, or Turbo C Compiler)
Item #35-6 **$29.95**

TO ORDER: Return the coupon on the next page or CALL TOLL-FREE 800-533-4372 (In CA 800-356-2002)

**** Also available at your local bookstore***

A Small-C Compiler: Language, Usage, Theory, and Design



by James E. Hendrix

This book contains a full presentation of the design and theory of the Small-C compiler and programming lanaguage. Full source code is included. In addition to a full, working Small-C compiler, this book provides an excellent example for learning basic compiler theory. Some of the features implemented include: recursive descent parsing, a one-pass algorithm, and the generation of assembly language code. You'll even learn how the compiler can be used to generate a new version of itself.

All sample programs are available on disk with full source code. A Microsoft or IBM Macro Assembler is necessary.

Book & Disk (MS-DOS)	**Item #97-6**	**$38.95**
Book	**Item #88-7**	**$23.95**

C Programming for MIDI

by Jim Conger

For musicians and programmers alike, **C Programming for MIDI** will help you create useful programs and libraries of software tools for music applications.

Author Jim Conger begins by outlining the features of MIDI (Musical Instrument Digital Interface) and its support of real-time access to musical devices. An introduction to C programming fundamentals as they relate to MIDI is also provided. The author fully demonstrates all these concepts with two MIDI applications: a patch librarian and a simple sequencer. Some of the fundamental MIDI programming elements you'll learn are:

- full development of a patch librarian program
- sequencing applications for the MPU-401 interface
- how to create screen displays
- how to write low-level assembly language routines for MIDI
- diagnostic tools for reviewing data
- menu selection
- terminals

All programs are available on disk with full source code. MS-DOS format. Supports both Microsoft C and Turbo C.

Book & Disk (MS-DOS)	**Item #90-9**	**$39.95**
Book	**Item #86-0**	**$22.95**

Turbo C:
The Art of Advanced Program Design, Optimization and Debugging

by Stephen R. Davis

Overflowing with example programs, this book fully describes the techniques necessary to skillfully program, optimize and debug in Turbo C. Every topic and Turbo C feature discussed is fully demonstrated in Turbo C source code examples. Advanced topics such as pointers; direct screen I/O; inline statements in Turbo C; and how to intercept and redirect BIOS calls are all covered in depth. The author further demonstrates these advanced topics by writing a RAM resident pop-up program in Turbo C. In addition the author fully outlines the differences between Unix C and Turbo C; the transition from Turbo Pascal to Turbo C; and the superset of K&R C features implemented in Turbo C and included in the proposed ANSI C standard.

Book & Disk (MS-DOS)	**Item #45-3**	**$39.95**
Book	**Item #38-0**	**$24.95**

TO ORDER: Return this ORDER FORM, along with your payment to M&T Books, 501 Galveston Drive, Redwood City, CA 94063. Or **CALL TOLL FREE 800-533-4372**, Mon. - Fri. 8 a.m. - 5 p.m. Pacific Standard Time. In CA CALL 800-356-2002.

Name ____________________

Address ____________________
(Please use street address, not P.O. Box)

City ____________ State ____ Zip ________

Day Phone ____________________
(In case we have questions about your order.)

☐ Check enclosed. Make payable to M&T Books.

☐ Charge my: ☐ Visa ☐ Master Card ☐ AmEx

Card # ____________ Exp. Date ________

Signature ____________________

Qty	Item #	Description	Price

Subtotal ________

Sales Tax - CA residents must add applicable sales tax ___% ________

Shipping and Handling ($2.95 per book) ________

Total ________

3139

EXAMINING ROOM

PC/Forms

Product:
PC/Forms, Version 1.21b
Target:
IBM PC, IBM PC AT, IBM PS/2, and compatibles
Requires:
DOS 2.0 or later; one floppy drive; 256K
Pricing:
C version $149.95; Turbo Pascal version $99.95
Vendor:
Golden Solution, P.O. Box 22216, Cleveland, OH 44122; 1 800-338-6754

Let's face it, screen layout and data validation are a drag. Most serious PC applications probably devote over half the code—and well over half the development effort—to tweaking displays and protecting the user from the GIGO syndrome.

Golden Solution's PC/Forms was designed to reduce this tedium and it does! In 15 minutes to half an hour, you can design a display, set up elaborate input validation criteria, test the form interactively, and generate the code to implement it. Writing and testing the equivalent source code from scratch could easily take several days.

The Turbo Pascal and C versions of PC/Forms are comprised of different sets of tools. We tested the C version, so that's what we'll talk about here.

The heart of the product is a stand-alone editor called FORMS. You run it to set up the display and validation and to generate the files used by your application. FORMS has bouncing-bar menus`a la Lotus 1-2-3. It's a highly visual environment, with windows popping up all over the place and pick lists and such, all of which have an intuitive feel. In layout mode, you paint your form on the display using function keys and the usual editing commands. There are commands for centering text, drawing boxes and lines, highlighting, rearranging, and so on.

When you're happy with the layout, you pick Attributes from the main menu, then cycle through the fields assigning validation parameters. Figure 1, below, shows the extent of the options available: picture, data type, decimal precision, mandatory response, and so on. A particularly intriguing option is aux edit, which ties a user-supplied routine to a field so that you can provide validation above and beyond the capabilities offered by PC/Forms.

The test selection from the utilities menu simulates a data entry session. You can step through the form, making sure the validation criteria work and that the order of fields is right. If not, you can jump back to the editing tools and fix it without leaving FORMS. This is a particularly handy feature.

When all is well, you generate two files. One is a .FRM file, a descriptor file for the form, which we'll discuss later. The other is an application shell in generic C. It's by no means a complete application, but it contains a data structure for the fields and all the code to load and execute the form (which is only about six lines). The shell is suitable for editing and insertion into a program as a function.

The other major component of the software is a header file and a runtime library. PCFORMS.H defines the data structures, function prototypes, and what not used by PC/Forms. You include it in your source program and link the object code with the runtime library.

There are actually three runtime libraries, one for each supported C compiler (Borland, Lattice, and Microsoft). You copy the one you need from the delivery diskette. They're

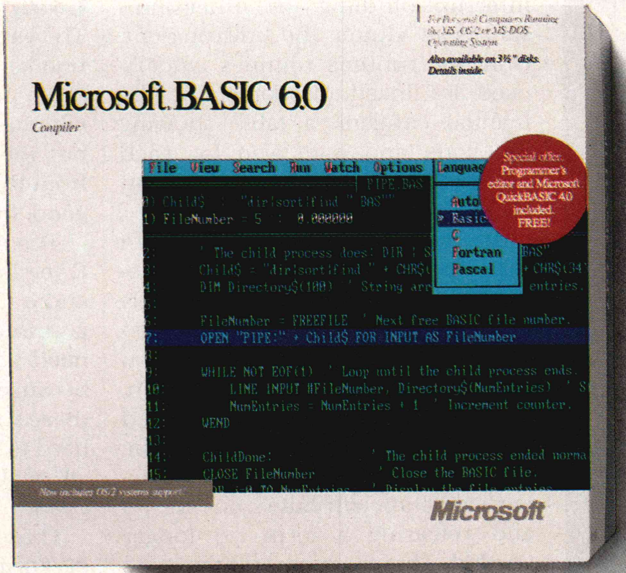
```
PC/Forms Version 1,2                           ©Copyright1987 Golden Software

  Single All
  Assign attributres to all fields

  Name: ADDRESS    Order;  4 Offset;    39 Rwo;  9 Col; 12 Len;      27

  Edit Mask;
  X
  Default Value;
  Auto Tab;           y   Right Justify;   N   Display Only;   N
  Must Respond;       N   Echo Data;       Y   Upper Case;     N
  Aux Edit Code;      0   Test Range;      N   Warning Only;   N
  Numeric Test;       N   Data Type;       S   Pecision;       0

           Lower Bound;       0,00    Upper Bound;   0,00

  [F1] Help  [F2] Display Field  [F3] Next Field  [F10] Return to Menu
```

Figure 1

Ron Copeland, associate editor for DDJ, is the coordinator for this review section. He welcomes your feedback on products worth reviewing.

NEW!

For pros. No cons.



Most BASICs are designed for people who only use BASIC part of the time.

Finally there's a new BASIC designed for professionals who spend their life with it. It's Microsoft® BASIC Compiler 6.0. It's got all the tools you need to make writing terrific code easier. Whether you want the implementation flexibility of MS-DOS® Or the advanced capabilities of OS/2.

With OS/2, you can develop large, sophisticated applications that go beyond the 640K barrier. Applications that take advantage of up to 16MB of RAM. And that exploit the potential of today's microprocessors.

Microsoft BASIC Compiler 6.0 has features you can work with, not around. Runtime library control helps you write the most compact manageable code. The customizable runtime library lets you add subroutines that can save disk space.

Since speed and hardware requirements vary, you can choose the floating-point math that suits you from among three math libraries. Including 8087/80287 coprocessor support, a floating point emulator and a fast math library for systems without math co-processors.

For the first time ever in a BASIC, you can get our popular advanced debugger, CodeView® Now you can sort out program logic at the source code level. Even if you happen to be using more than one language.

And to get your DOS projects off to a quick start, Microsoft BASIC Compiler 6.0 includes Microsoft QuickBasic 4.0.

Call (800) 541-1261, Dept. D85, and we'll give you the name of your nearest Microsoft dealer. Because at 6.0, BASIC has finally come of age.

OS/2 Features
- Contains all of the OS/2 utilities to make protected-mode as well as real mode applications.
- OS/2 language additions such as SHELL, OPEN PIPE, ON SIGNAL and SLEEP.
- First programmer's editor that lets you write both OS/2 and MS-DOS programs.

CodeView Features
- Trace execution, set breakpoints, set watch variables, view source code, see data automatically updated as variables are displayed.

BASIC language features
- Fastest stand alone, multiple module executable with no runtime fees.
- User defined event trapping.
- Multiple module error handling.
- Supports records, recursion, huge arrays.
- Additional data types—4 byte integers, fixed length strings.
- Supports all major structured programming constructs including SELECT CASE and FUNCTION procedures.

Microsoft®

(continued from page 122)

all small models. Golden Software includes source code for the runtime system, so if you need a different model, you can recompile appropriately.

Everything your application needs in order to use PC/Forms is linked into the .EXE file. There's no separate runtime support package, TSR, interrupt-diddling, or other nuisances to clutter the environment. Only the runtime routines actually used are linked, of course, and the manual contains a table showing the code and data size for each routine. The average size is about 1K in a range of 14 to 3,974 bytes.

All identifiers have the form pcf_fname, where fname is something like "display_form" or "error." The pcf prefix makes them distinctive. It takes about half a dozen functions to load and display a form, initialize it, and get the validated input. Other functions among the 17 available do things such as altering validation attributes on the fly and releasing a form no longer needed. There are an additional 23 functions for such things as video and string management.

The .FRM descriptor produced by the FORMS editor is an ASCII file containing information specific to a given form. The runtime system needs it to implement the form and perform validation. Opening a form is a matter of loading this file. You can have several forms open at one time, and any given form can span up to ten pages (display panels). But watch out: the forms go on the stack, and you'll need a mighty big stack if you have several open at once. The FORMS editor has a utility that sizes a form and tells you how much stack space it will require.

The crucial runtime function is pcf_get_form(). When called with a form displayed, it manages user input and validation. The results are placed in a data structure corresponding to the form, whose fields can have user-assigned names. Your program then fetches data from the structure and does its thing with it. This makes the data entry portion of a loop almost ridiculously simple:

```
do {
    pcf_display_form (name, page);
    pcf_clear_form_buffer(&buf, defaults);
    pcf_put_form (&buffer);
    pcf_get_form (&buffer, &Term);
    /* then do data processing */
    while (some_condition);
```

Complaint Department

The vendor ought to include a function key template. Each function key has a purpose and some have an Alt command as well. I finally printed out the function key layout screen from the help system (which is very good, by the way).

The manual needs work. With the C version, you get the Pascal manual and a C addendum. The addendum is printed on yellow paper and you need a magnifying glass to read it because the print is so small. And there's no index, an omission that's hard to forgive even though the overall quality of the documentation is good.

These quibbles notwithstanding, PC/Forms is a real gem. It can truly save countless hours of programming, which makes it a contribution to productivity that will pay for itself many times over.

by Kent Porter

DE

Product:
DE, Version 1.2
Target:
IBM PC, IBM AT, IBM PS/2, and compatibles
Requires:
One floppy; 256K
Pricing:
$75
Vendor:
David Livshin
26 Niles Rd.
Randolph, MA 02368
617-986-7491

I don't know what DE stands for, the manual doesn't say. Were I to hazard a guess, however, I'd say it means "deluxe EMACS."

DE is a stretched version of the

(continued from page 124)

standard EMACS editor. It delivers a host of impressive features that make it a macro-programmable, customizable editor with an unlimited number of overlapping and/or tiled windows.

Unlike competing editors such as Brief, DE doesn't require a bunch of support files. It comes as a single 69K .EXE file on the delivery diskette, and installation is as simple as copying that file to your hard disk or work floppy. To invoke the editor, just type DE. Up to two command-line arguments are allowed: a -NOBAK switch to tell the program not to make a backup copy of the edited file(s), and the name of a file to be edited.

If you want to pull several files into different windows, you can fetch them after DE is up and running. The command ^X^V causes the program to prompt for a filename, then creates a new window and loads the file into it. You keep doing this until all the files you want are loaded.

Initially the windows are tiled. Each one has an information line at the bottom showing the associated filename, cursor position, number of lines, and so on, and most significantly, the window number. Various keystroke combinations let you move sequentially forward and backward among windows or jump directly from one to another. Other commands resize and drag windows so that they overlap like pieces of paper: the now familiar desktop metaphor. In overlapping mode, the current window is always on top.

If you don't like the hierarchy of windows, you can change the sequence numbering. This is a nice touch. You can make the modules you're working on neighbors in the hierarchy; it takes fewer keystrokes to move among adjacent windows than to make jumps.

The command ^X 1 does a thing called zoom and rise to the currently selected window. This changes the operation of DE by expanding each window to full-screen size and placing the current window on top. Oddly, the jump-to-next and jump-to-previous commands (^X n and ^X p) don't work any more in this mode; you have to jump to a specific window. And there's no way that I found to undo the zoom-and-rise mode. Once you're in it, you're there to stay.

EMACS commands in general are less than intuitive, and DE continues the tradition by adding still more to the repertoire. All DE commands except those dealing with editing and cursor movement begin with either ESC or ^X, followed by a keystroke denoting the command. Some make sense (^X^I to insert a file, ^X^W to write to a file, and so on), but most have no discernible connection with anything. Examples are ^__ to invoke the DOS shell, and ^X z and ^X^Z to enlarge and shrink a window, respectively.

Consequently, the vendor in-

(continued from page 127)

cludes a cheat sheet showing all the keystroke commands. There's also a limited help function: type ESC ^A, then a keystroke combination, and DE tells you what function the combination performs.

Each keyboard command is mapped to a DE macro through what the vendor calls "default binding." For example, ^X^S is bound to the macro w_cfil, which writes to the current file. This association of keystrokes to macros opens the way to two features of DE: customization of the keyboard and programmability.

DE comes with 84 different macros, of which 73 are bound to default keystrokes and the other 11 (all of them related to window management) are unassigned. If you don't like the default bindings, or you want to add some bindings of your own, the command ESC ^@ runs an embedded utility that maps keystrokes to macros.

You can build your own more complex macros by combining those built into DE, thus creating editor programs invoked by a keystroke. Seldom needed programs can be stored in separate ASCII files and run with the ESC e command, which asks for the filename, loads it, and treats the contents as commands. Unfortunately, the manual barely glances at this useful feature.

Also alluded to but never explained in the manual is something called the DE.INI file. It presumably contains initialization commands that permanently map macros to keystrokes and perform other fixed set-up tasks.

It's a pity that the DE manual is not up to the quality of the software it purports to describe. A slim 26 pages, it contains terse descriptions of the macros, a little about windows and commands, and not much of anything else. The author assumes that you already know EMACS, and so leaves it to your imagination how to use the editor and its features. There isn't even a hint of a tutorial. The best part of the documentation is the cheat sheet, which puts most of the manual's contents on a card providing at a glance reference.

Overall, DE is a good editor with a lot of capability per buck.

by Kent Porter

Soft-ICE

Product:
Soft-ICE, Version 1.01
Target:
80386-based MS-DOS computers
Requires:
DOS 2.0 or later; AT BIOS
Pricing:
$386
Vendor:
Nu-Mega Technologies
P.O. Box 7607
Nashua, NH 03060-7607
603-888-2386

Soft-ICE is a product any MS-DOS developer serious enough to own a 386 machine should have. As the name implies, it provides the capabilities of an in-circuit emulator via software. For those of you not familiar with in-circuit emulators, a brief description is in order.

An in-circuit emulator (ICE) is a tool that replaces the CPU in a microprocessor-based product with a "pod" that plugs into the CPU's socket. This pod is normally connected to a box containing a control computer and some special hardware. The special hardware is used to detect user-specified conditions and to stop the processor when they occur. Another feature commonly found in ICEs is trace memory, so that when the processor stops, you can see where it has been recently. ICEs are normally expensive, and often designed more for debugging hardware rather than debugging software.

Soft-ICE gives 386 owners all of this capability, except trace memory, when debugging MS-DOS programs. It does this by using special features of the 386 normally used in writing operating systems (see February, 1988 *DDJ* for more details). Note that you can't use Soft-ICE to debug protected mode programs.

Soft-ICE can be used either stand alone or in conjunction with your favorite debugger. As a stand-alone debugger, it includes all of the necessary commands to disassemble, dump, and edit memory; to display and change registers; to peek and poke at I/O ports; and to manage breakpoints. A very useful help facility is also included, as well as a command to display the DOS system memory map. As you type commands, Soft-ICE displays a list of options.

Getting all the Breaks

Perhaps more than any other debugger, Soft-ICE lets you control breakpoints. You can set breakpoints to occur when any byte, word, or double word is read, written, read or written, or executed. For read/write breakpoints, you can include a qualifying value that must be matched or not matched. Breakpoints may be configured against a larger address range such that a breakpoint occurs on any read, write, or read or write in the range. I/O port accesses can cause breakpoints, qualified by values if desired. Execution of either a hardware or software interrupt (qualified by a value in AL, AH, or AX if desired) can cause a breakpoint. Of course, each of these breakpoints may be qualified with a count so that you can ignore the first 100 times you do any of them. You can even combine breakpoints so that a breakpoint only occurs after all of the selected breakpoints have individually occurred. Finally, breakpoints may be qualified with the location of the breaking instruction, to guarantee that the instruction is either inside or outside a range. All in all, a very comprehensive set of breakpoint capabilities, all of which can be used either stand alone or

(continued from page 129)

with another debugger.

If all of this isn't quite enough, you can configure Soft-ICE to generate a software interrupt when it detects a breakpoint. This user-provided interrupt gets all of the registers as they were when Soft-ICE got control, allowing the interrupt handler to do anything it likes.

When used with another debugger, Soft-ICE can be configured to trigger the other debugger when a Soft-ICE detected breakpoint occurs. It can cause an interrupt 1 or 3, or an NMI. Soft-ICE normally passes interrupt 3 onto whatever awaits it, but it can also be configured so that an INT3 returns control to Soft-ICE.

Soft-ICE is very flexible. It may be installed in normal memory, in extended memory, or in COMPAQ extended memory. If Soft-ICE finds extended memory it will automatically load there requiring none of the lower 640K. In a machine with only 640K, the program demands between 56 to 60K, rendering this segment 60K invisible to DOS.

Soft-ICE can share extended memory with other drivers, such as VDISK or RAMDRIVE. It emulates the necessary parts of the LOADALL instruction for RAMDRIVE. You can change the keystrokes used to invoke Soft-ICE. It can even boot up stand-alone code and debug it because Soft-ICE doesn't require any DOS services. Soft-ICE can also be used to debug MS-DOS device drivers. And, joy of joys, you can even debug interrupt handlers (including the keyboard interrupt) while Soft-ICE is using it.

The documentation is solid and tells you everything you need to know in about 100 pages. A tutorial chapter takes you through debugging, a simple program and is highly recommended for both beginners and those already experienced in debugging. [It does however require an IBM or equivalent for the BIOS routines. —ED]

All in all, Soft-ICE is an excellent tool for debugging 8086 programs. Compared to a true in-line emulator, (even if you have to buy a 386 machine to run it on) it's cheaper and provides superior breakpoint facilities; the only thing missing is a trace memory.

by Richard Relph

DDJ

Object-Oriented Engineering

It can mean an end to user interface problems.

Creating a user interface that has the right 'look and feel' for your application can be a monster of a job.

And doing it so your application runs consistently on many different platforms can be downright horrifying.

It involves writing lots of tedious, low level code over and over again. Time more profitably spent working on the application itself.

Solving this problem is simple.

Get Stepstone's ICpak™ 201.

It lets you quickly and easily build multi-window iconic user interfaces that operate without modification across a wide variety of workstations in both monochrome and color.

ICpak 201 is a set of over 50 pretested Software-IC®s based on our advanced object-oriented technology. It's fast to learn and has been thoroughly proven in the real world.

Compatibility is no problem. ICpak 201 lets your application run on all popular workstations and graphic substrates, including X-Windows.™

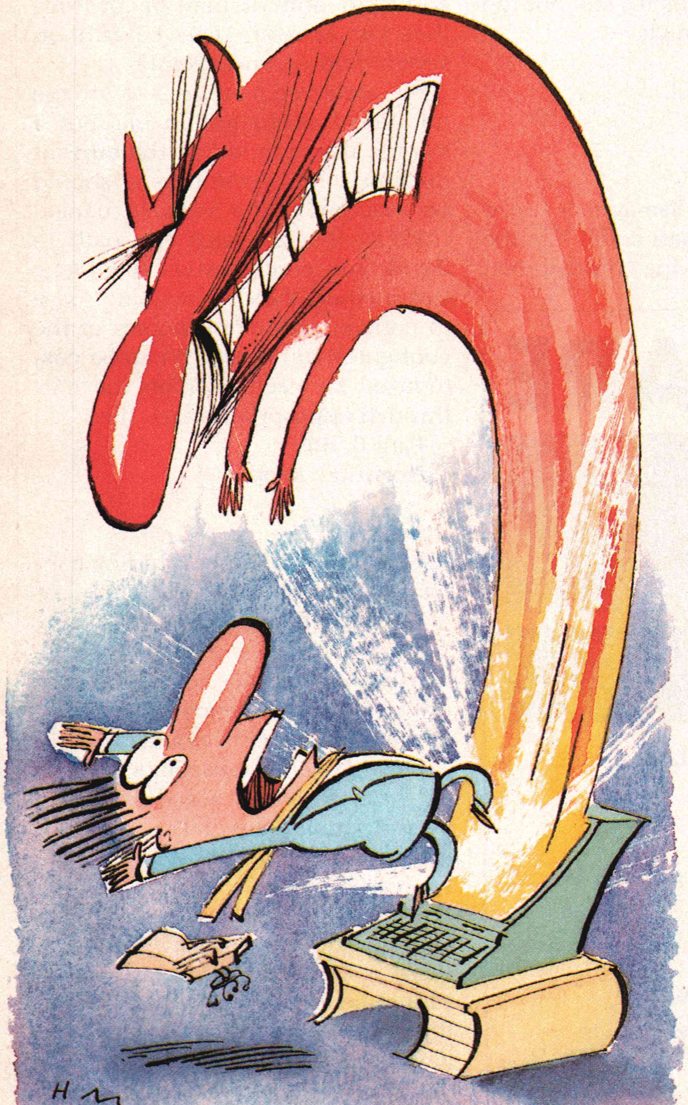
Prototyping is a snap. You can design, test and refine your user interfaces independent of your applications.

ICpak 201 comes with a rich set of features: icon editor, unlimited windows, unlimited fonts, and more.

Just as important is our technical support. We're not just the distributors. We're the developers. So we'll be able to answer any questions.

And you can count on us to keep you supplied with Software-ICs that deliver the best in user interface technology.

So look into ICpak 201. It may be just what you need to tame the user interface monster forever.



Stepstone™
The Leader in Object-Oriented Technology

The Stepstone Corporation 75 Glen Road Sandy Hook, CT 06482 203 426 1875 Telex 506127 FAX 203 270 0106

CIRCLE NO. 231 ON READER SERVICE CARD

(continued from page 14)

Compatibility Standards

Dear *DDJ*,

I've been thinking lately about how unfortunate it is that there is so little standardization in the computer industry, making it impossible to run programs written for one machine on another. I have an idea that could alleviate some of the compatibility problems, and I'd be interested in readers' responses to it.

The idea is this: Design a standard intermediate language, similar to a language that would be generated by the syntax-analysis pass of a compiler (for input to the code generator). Then, use this new language as the form in which software is distributed for use on computers, instead of as native code that is specific to a particular system. The program loader on each computer would recognize the special intermediate-language file and invoke a fast code generator to translate it to native code prior to beginning execution.

There are problems with this—one being that different systems have different capabilities and different hardware. Most systems, however, have a display device that can display ASCII characters and a printer. For many programs this is all that is needed. The intermediate language could include instructions to determine the capabilities of the system (for example, display size), allowing the program to adapt to different machines.

Another problem is that the program distribution medium is different for different systems. Even computers that use floppy disks typically each have their own disk format. This is indeed unfortunate, and manufacturers should be severely taken to task for not standardizing disk format. Perhaps it's still not too late to do this, though.

Ted Toal
Nevada City, Calif.

LAN Security

Dear *DDJ*,

It was with some dismay that I read Allen Holub's C Chest column in the February issue. Saving configuration information in the .EXE file is the easiest way I know of to discourage the use of a program on a local-area network. If users must be able to write to the .EXE file to use the program, then the program becomes a hole in system security, an opening for Trojan horse programs. Second, if users need to write to the .EXE file, multiple instances of the program cannot be run at the same time.

Often a program that has not been written with the LAN environment in mind may be used on a LAN with no modification, provided that the program may be configured dynamically. If the program may be run when its file attributes are shareable and read only, it may be possible that the program can be run on several nodes concurrently.

My preferred method is to use the traditional configuration file, but if the file is not found in the current directory, the environment should be searched for a variable named *CONFIG* that contains the path to the default configuration file. Using this method the disk does not have to contain redundant copies of the configuration file, but it can be customized for each user on the LAN through appropriate batch files.

Paul B. Hill
Norwood, Mass.

Dear *DDJ*,

Allen Holub's method of hiding configuration information in an application's .EXE file (C Chest, February 1988) is elegant and quite instructive, but I would like to raise a point or two in favor of keeping configuration information in separate files.

Most shared applications on a local-area network are located in shared public directories that are accessed via the DOS path or an equivalent network function. Users of the shared applications are usually granted read-only access to these public directories. Most users cannot be allowed to modify files in the public areas as this would be an invitation to disaster. Application programs that write into their own .EXE file do not work well under these circumstances and can cause LAN managers severe headaches. The re-

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox** (DOS,OS/2).

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.

Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

"Platform" — Code Quality

Sun,Apollo,SGI	— 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2	— 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS	— no competitors, since November, 1986.
286, 386 UNIX	— 66% better than pcc (Dhrystone, 386).
VAX VMS	— ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix	— 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd	— 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX	— much better than any C, and VS Pascal.
AMD 29000	— >40,000 Dhrystones! Available in Q2, cross.

(408) 429-6382, telex 493-0879. ***Since 1979.***

Meta Ware™
INCORPORATED
903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.

CIRCLE NO. 158 ON READER SERVICE CARD

(continued from page 132)

sult is that you end up with several complete copies of the application program scattered about.

It is possible for a LAN manager to arrange for the proper storage of an application's configuration file if the program is designed with some thought to the problem. I believe the best way to handle the location of the configuration file is for the application to accept a command-line parameter that points out the complete path where the configuration file is located.

The LAN manager usually has methods of setting environment variables when users log on to a LAN that can specify user names of physical workstation numbers. These environment variables can then be used with shared public batch files to execute the application and specify the appropriate configuration file for a particular user or workstation. It is useful to be able to specify a configuration file in association with a physical workstation because most networks have a variety of workstation hardware.

The programs that are most convenient to install and use on local-area networks are those that keep things simple. The best applications are often those that consist of a single .EXE file with no other external files. These are also the programs that LAN managers will purchase in quantity rather than those applications that cause difficulties in configuration, installation, and maintenance. Most of the software currently used on LANs are not the large expensive packages that do strange tricks with configuration and security information but the same simple single-user products that work so well on Personal Computers.

Phillip M. Nickell
Longmont, Colo.

Was He Misguided?

Dear *DDJ*,

I was happy to see the review of The Norton Guides (Examining Room, February 1988); however, I was reminded of these words by Lewis Mumford in the *Pentagon of Power*: "Unfortunately, 'information retrieving,' however swift, is no substitute for discovering by direct personal inspection knowledge whose very existence one had possibly never been aware of, and following it at one's own pace through the further ramification of relevant literature. But even if books are not abandoned, but continue their present rate of production, the multiplication of microfilms actually magnifies the central problem—that of coping with quantity—and postpones the real solution, which must be conceived on quite other than purely mechanical lines: namely, by a reassertion of human selectivity and moral self-discipline, leading to continent productivity. Without such self-imposed restraints the over-production of books will bring about a state of intellectual enervation and depletion hardly to be distinguished from massive ignorance."

As a reasonably happy owner of The Guides, I eagerly dug into the review, which upon reading, I'm afraid to say seemed like a cursory tidbit that I had not expected to find in *DDJ*. I feel there are three major problems with it:

1. I originally thought that the 37K memory residency TSR requirement was a gnomish munchie, but after looking at the 38K file size on my floppy, I thought there was definitely a mistake here. I have run this program on both an IBM PC and an IBM AT compatible with PC-DOS/MS-DOS, and it takes up 71,920 bytes installed.
2. One of the original problems with the advertising, and especially on the box my program came in, was the reference to being able to run it on a floppy-disk-based system. This is possible if you have drives that hold more than the 600K needed for either the MASM or C database. A hard disk is probably required rather than recommended, unless you wish to write your own or just use a BASIC database. I was not pleased to find that the assembly-language database would not run on my IBM PC floppy system. The review made no mention of this possible predicament.
3. One of the most useful aspects of the program is the capability of The Guides (given the right active menu) when initially activated to do an automatic lookup of the entry for the word by the cursor. It's a handy feature to have enabled but nary a mention of it in the review.

I would think that if you're going to do these reviews, you need to have people to do them who have more than a passing interest in the material. As the ultimate end-user of some of this software, I'd like to get information other than what I can get from ads and the users' booklet. I'd also like to see how the product compares with others that purport to do the same thing but cost less.

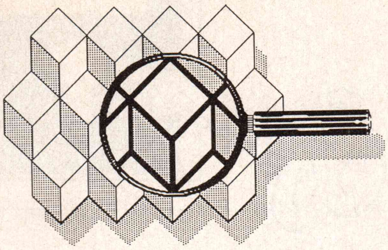
Richard L. Henley
via CompuServe

Kent Porter responds:
Let's go by the numbers:

1. We're both wrong according to CHKDSK; it takes 72,032 bytes on my AT. The NG.EXE file is 37K, not RAM resident.
2. Good point. I haven't run it on a floppy-based system, but Richard's point makes sense because the databases are large.
3. Auto-lookup is indeed a dandy feature. If it got short shrift, it's because space in these reviews is limited.

DDJ

OF INTEREST



Products for Developers

Dan Bricklin's Demo II Program is now available from **Software Garden**. This new version comes with a 220 page manual, keyboard templates, an on-line tutorial, and sample files. New features include the ability to capture bitmapped graphics images from other programs, string and numeric variables, and a run facility with over 100 new actions to execute while running. Demo II also comes with a license to make an unlimited number of copies of the runtime. The product runs on 512K IBM PC, IBM PC AT, IBM PS/2 or compatibles using DOS 2.0 or later. A monochrome display adapter, CGA, EGA, VGA, Hercules Graphics Card or the equivalent is also required. Demo II sells for $195. Reader Service No. 20.
Software Garden Inc.
P.O. Box 373
Newton Highlands, MA 02161
617-332-2240

National Design Inc. (NDI) has released Genesis 1024 and Genesis 1280 intelligent PC color-graphics controllers bundled with an implementation of the Computer Graphics Interface (CGI) developed by Nova Graphics International and Metagraphics' MetaWINDOW.

The NOVA*CGI provides software developers a CGI interface that is now resident on the NDI controllers. By executing the CGI on the NDI board, the developer can perform graphics routines 20 to 40 times faster than on a PC using an EGA card.

The Genesis 1024 (a 640 × 480 up to 1,024 × 768, 16 color card), and the Genesis 1280 (a 640 × 480 up to 1,280 × 1,024, 256 color card) use Texas Instruments' 34010 graphics system processor operating at 40 or 50 MHz. The Genesis products also provide expandable memory up to 32 Mbyte.

The Genesis 1024 costs $1,700 and the price of the Genesis 1280 ranges from $2,995 to $3,995, depending on configuration. Reader Service No. 21.
National Design Inc.
9171 Capital of Texas Hwy. N.
Houston Bldg., Ste. 230
Austin, TX 78759
512-343-5055

The C Programming Language, Second Edition by Brian W. Kernighan and Dennis M. Ritchie has been published by **Prentice Hall**. This new edition is based on the draft-proposed ANSI C Standard. The book makes precise the features that were not spelled out in the original definition of C, and states explicitly which aspects of the language remain machine dependent. New features from the ANSI standard, such as function prototypes and the standard library, are also explained. Additional changes in the new edition include a C reference manual, an appendix describing the standard library, and an appendix summarizing changes between the first edition and the draft-proposed ANSI standard. The price of the book is $40 for cloth and $28 for paper. Prentice Hall will continue to publish the first edition. Reader Service No. 22.
Prentice Hall
Prentice Hall Bldg.
Englewood Cliffs, NJ 07632
201-592-2000

The Software Link (TSL) has announced its newly formed Developer Relations program. The program is based around the company's newly released PC-MOS/386 Technical Reference Manual. Participants subscribe for an annual fee of $500 to development support service which includes: PC-MOS/386 Technical Reference Manual and updates as they become available, access to TSL's support line, upgrades of PC-MOS/386, participation in TSL's product certification program and inclusion in TSL's product reference guide, and purchase of one PC-MOS package for development use at a reduced rate. Reader Service No. 23.
The Software Link
3577 Parkway Ln.
Norcross, GA 30092
404-448-5465

World Wide Data has released Charm, a C source application generator. Charm is an integrated application generator for Unix and VMS environments that automatically creates fully documented C source code. Charm's 4GL, dali (data access language interface) is a natural extension of the interactive screen and program generator. All the standard field default and verification options in Charm are dali programs. Reader Service No. 24.
World Wide Data
17 Battery Pl.
New York, NY 10004
718-438-2807

The Renaissance Graphics Device Interface (RGDI) Developer's Kit is a toolkit for software developers that enables them to develop graphics applications that take advantage of the speed of a special graphics processing chip. **Renaissance GRX's** new product includes: Rendition I Advanced Graphics Controller incorporating RGDI, Rendition I user's guide, RGDI programmer's technical reference manual, TI 34010 user's guide, and development software.

The RGDI is a graphics controller interface that allows a software program to send messages to the Texas Instruments TMS34010 Graphics System Processor, a 32-bit, high-speed integrated circuit that is optimized for graphics performance.

For developers who wish to write in assembly language, Renaissance offers an optional accompanying advanced toolkit which includes: TI 34010 debugger and user's guide; TI 34010 assembler package, including an assembler, linker, and simulator; and development utilities.

For developers wishing EGA compatibility, an optional Rendition EGA

READER SERVICE

Knowing a little more about you helps us to tailor *Dr. Dobb's Journal* to specific reader interests. Please take a moment to answer the questions below before you return this card for free product information. Thanks very much!

A. Do you design or develop software for professional applications? 1. ☐ Yes 2. ☐ No

B. What type of computer(s) do you use at work? Please check all that apply.

1. ☐ IBM PC or XT
2. ☐ IBM AT
3. ☐ IBM PS/2 (all models)
4. ☐ Other IBM
5. ☐ PC or XT compatible
6. ☐ AT compatible
7. ☐ 80386 computer
8. ☐ Macintosh Plus
9. ☐ Macintosh II
10. ☐ Macintosh SE
11. ☐ Sun
12. ☐ Apollo II
13. ☐ CP/M, TRS-80, or Apple
14. ☐ Other

C. Please estimate the total amount you anticipate spending on new hardware and software (through approval, recommendation, or specifying) in the next 12 months:

1. ☐ $1000 or less
2. ☐ $1000 to 2500
3. ☐ $2500 to 5000
4. ☐ $5000 to 10,000
5. ☐ $10,000 or more

D. Does your organization buy *hardware* from mail order retailers?
1. ☐ Seldom 2. ☐ Occasionally 3. ☐ Frequently

E. Does your organization buy *software* from mail order retailers?
1. ☐ Seldom 2. ☐ Occasionally 3. ☐ Frequently

Please indicate how often you read the following columns:

F. Editorial:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
G. Running Light:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
H. Archives:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
I. Letters:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
J. C Chest:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
K. The Forth Column	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
L. To the Macs:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
M. Structured Programming:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
N. Artificial Intelligence:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
O. Of Interest:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
P. Swaine's Flames:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never

Q. What action have you taken as a result of seeing the advertisements in *Dr. Dobb's Journal*?

1. ☐ Purchased products
2. ☐ Recommended purchase
3. ☐ Requested more information
4. ☐ Called advertiser
5. ☐ No action

R. Please indicate which of the following languages you use professionally:

1. ☐ Assembler
2. ☐ C
3. ☐ BASIC
4. ☐ Pascal
5. ☐ dBase
6. ☐ Fortran
7. ☐ 4GLS
8. ☐ Cobol
9. ☐ LISP
10. ☐ Prolog
11. ☐ Forth
12. ☐ ADA
13. ☐ Modula-2
14. ☐ APL
15. ☐ Smalltalk
16. ☐ Other

S. Which of the following operating systems do you use professionally?

1. ☐ MS-DOS or PC-DOS
2. ☐ OS/2
3. ☐ Macintosh Finder
4. ☐ Unix or Xenix
5. ☐ Real-time operating systems
6. ☐ Other

. . . FOR FAST, FREE INFO,

use this card to find out about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________

Title ______________________

Company ______________ **Phone** (____) ______________

Address ______________________

City/State/Zip ______________________

1	21	41	61	81	101	121	141	161	181	201	221	241	261	281
2	22	42	62	82	102	122	142	162	182	202	222	242	262	282
3	23	43	63	83	103	123	143	163	183	203	223	243	263	283
4	24	44	64	84	104	124	144	164	184	204	224	244	264	284
5	25	45	65	85	105	125	145	165	185	205	225	245	265	285
6	26	46	66	86	106	126	146	166	186	206	226	246	266	286
7	27	47	67	87	107	127	147	167	187	207	227	247	267	287
8	28	48	68	88	108	128	148	168	188	208	228	248	268	288
9	29	49	69	89	109	129	149	169	189	209	229	249	269	289
10	30	50	70	90	110	130	150	170	190	210	230	250	270	290
11	31	51	71	91	111	131	151	171	191	211	231	251	271	291
12	32	52	72	92	112	132	152	172	192	212	232	252	272	292
13	33	53	73	93	113	133	153	173	193	213	233	253	273	293
14	34	54	74	94	114	134	154	174	194	214	234	254	274	294
15	35	55	75	95	115	135	155	175	195	215	235	255	275	295
16	36	56	76	96	116	136	156	176	196	216	236	256	276	296
17	37	57	77	97	117	137	157	177	197	217	237	257	277	297
18	38	58	78	98	118	138	158	178	198	218	238	258	278	298
19	39	59	79	99	119	139	159	179	199	219	239	259	279	298
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300

May '88: use before August 31, 1988

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

READER SERVICE

Knowing a little more about you helps us to tailor *Dr. Dobb's Journal* to specific reader interests. Please take a moment to answer the questions below before you return this card for free product information. Thanks very much!

A. Do you design or develop software for professional applications? 1. ☐ Yes 2. ☐ No

B. What type of computer(s) do you use at work? Please check all that apply.

1. ☐ IBM PC or XT
2. ☐ IBM AT
3. ☐ IBM PS/2 (all models)
4. ☐ Other IBM
5. ☐ PC or XT compatible
6. ☐ AT compatible
7. ☐ 80386 computer
8. ☐ Macintosh Plus
9. ☐ Macintosh II
10. ☐ Macintosh SE
11. ☐ Sun
12. ☐ Apollo II
13. ☐ CP/M, TRS-80, or Apple
14. ☐ Other

C. Please estimate the total amount you anticipate spending on new hardware and software (through approval, recommendation, or specifying) in the next 12 months:

1. ☐ $1000 or less
2. ☐ $1000 to 2500
3. ☐ $2500 to 5000
4. ☐ $5000 to 10,000
5. ☐ $10,000 or more

D. Does your organization buy *hardware* from mail order retailers?
1. ☐ Seldom 2. ☐ Occasionally 3. ☐ Frequently

E. Does your organization buy *software* from mail order retailers?
1. ☐ Seldom 2. ☐ Occasionally 3. ☐ Frequently

Please indicate how often you read the following columns:

F. Editorial:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
G. Running Light:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
H. Archives:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
I. Letters:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
J. C Chest:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
K. The Forth Column	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
L. To the Macs:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
M. Structured Programming:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
N. Artificial Intelligence:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
O. Of Interest:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never
P. Swaine's Flames:	1. ☐ Always	2. ☐ Occasionally	3. ☐ Never

Q. What action have you taken as a result of seeing the advertisements in *Dr. Dobb's Journal*?

1. ☐ Purchased products
2. ☐ Recommended purchase
3. ☐ Requested more information
4. ☐ Called advertiser
5. ☐ No action

R. Please indicate which of the following languages you use professionally:

1. ☐ Assembler
2. ☐ C
3. ☐ BASIC
4. ☐ Pascal
5. ☐ dBase
6. ☐ Fortran
7. ☐ 4GLS
8. ☐ Cobol
9. ☐ LISP
10. ☐ Prolog
11. ☐ Forth
12. ☐ ADA
13. ☐ Modula-2
14. ☐ APL
15. ☐ Smalltalk
16. ☐ Other

S. Which of the following operating systems do you use professionally?

1. ☐ MS-DOS or PC-DOS
2. ☐ OS/2
3. ☐ Macintosh Finder
4. ☐ Unix or Xenix
5. ☐ Real-time operating systems
6. ☐ Other

. . . FOR FAST, FREE INFO,

use this card to find out about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________

Title ______________________

Company ______________ **Phone** (____) ______________

Address ______________________

City/State/Zip ______________________

1	21	41	61	81	101	121	141	161	181	201	221	241	261	281
2	22	42	62	82	102	122	142	162	182	202	222	242	262	282
3	23	43	63	83	103	123	143	163	183	203	223	243	263	283
4	24	44	64	84	104	124	144	164	184	204	224	244	264	284
5	25	45	65	85	105	125	145	165	185	205	225	245	265	285
6	26	46	66	86	106	126	146	166	186	206	226	246	266	286
7	27	47	67	87	107	127	147	167	187	207	227	247	267	287
8	28	48	68	88	108	128	148	168	188	208	228	248	268	288
9	29	49	69	89	109	129	149	169	189	209	229	249	269	289
10	30	50	70	90	110	130	150	170	190	210	230	250	270	290
11	31	51	71	91	111	131	151	171	191	211	231	251	271	291
12	32	52	72	92	112	132	152	172	192	212	232	252	272	292
13	33	53	73	93	113	133	153	173	193	213	233	253	273	293
14	34	54	74	94	114	134	154	174	194	214	234	254	274	294
15	35	55	75	95	115	135	155	175	195	215	235	255	275	295
16	36	56	76	96	116	136	156	176	196	216	236	256	276	296
17	37	57	77	97	117	137	157	177	197	217	237	257	277	297
18	38	58	78	98	118	138	158	178	198	218	238	258	278	298
19	39	59	79	99	119	139	159	179	199	219	239	259	279	298
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300

May '88: use before August 31, 1988

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT 200 PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Reader Service Management Dept.
P. O. Box 5282
Pittsfield, MA 01203-9925

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT 200 PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Reader Service Management Dept.
P. O. Box 5282
Pittsfield, MA 01203-9925

The Advertiser Index

Advertiser Name	Page #	RS#
Abraxas Software	70	75
Aesoft	132	77
AI Architects	32	79
Aker Corporation	44	*
American Cybernetics	139	81
Andsor Research Inc.	98	83
Apollo Computer, Inc.	21-23	*
Arity Corporation	25	85
Ashton Tate	41	87
Austin Code Works	80	89
Basis Incorporated	116	90
Blaise Computing	2	91
Block Island Tech.	129	93
Borland International	1	95
Breakpoint Computer Systems, Inc.	67	97
Burton Systems Software	61	*
C Users Group	61	99
CAE/SAR Systems, Inc.	45	101
Clarion Software	42	103
Coders Source (The)	128	237
CNS, Inc.	71	105
Cobalt Blue	86	107
Compaq Computer Corporation	16-17	*
Compu View	13	109
Creative Programming	79	*
Crosstalk Communications	C6	111
Datalight	52	113
DDJ Back Issues	119	*
DDJ Subscriptions	64	*
Digitalk	38-39	115
Disk Software	67	117
Ecosoft, Inc.	66	119
Elan Computer Group	70	121
Elan Computer Group	103	122
Entelekon Software Systems	119	124
Essential Software	94	126
Fair-Com	118	128
Franz, Inc.	83	130
Gimpel Software	36	*
Golden Bow Systems	104	132
Golden Solutions	35	134
Greenleaf Software	125	136
IET/Inference Engine Technologies	91	138
John Wiley & Sons	101	140
Lattice, Inc.	34	142
Lugaru Software Ltd.	84	144
M/SJ Subscriptions	96	*
M&T Catalog of Books & Software Tools	120-121	*
Machine Independent Software Corp.	68	146
Magna Carta Software	69	148
Manx Software Systems	7	150
Media Cybernetics	29	152
Meridian Software Systems	53	154
Metagraphics Software Corporation	106	156
MetaWare Incorporated	133	158
Micro Edge	43	160
Micro Way	75	162
Microsoft	107	*
Microsoft	123	*
Microsoft	C5	*
Microsoft University	112	*
Migent Software	4-5	164
Mix Software	126	166
Mortice Kern Systems, Inc.	74	168
Nanosoft Associates	140	170
Norton Utilities (The)	115	172
Nu-Mega Technologies	87	174
NWP Intelligent Solutions, Inc.	81	176
Oakland Group, Inc. (The)	37	178
Oasys	28	180
Pathfinder Software	138	182
Peacock Systems	130	184
Phar Lap Software, Inc.	90	186
PMI	95	188
Polytron Corporation	15	190
Programmer's Connection	142-143	192
Programmer's Paradise	10	194
Programmer's Paradise	11	195
Programmer's Shop (The)	88-89	197-199
QCAD Systems, Inc.	86	200
Quadbase Systems Inc.	84	202
Quarterdeck Office Systems	30-31	204
R.J. Swantek	105	206
Raima Corporation	48	*
SAS Institute	93	*
Scientific Endeavors	114	208
Secom Information Products Co.	97	210
Semi-Disk Systems	111	212
Sharpe Systems Corporation	135	214
SLR Systems	50	216
Softfocus	127	218
Software Connections Inc.	102	220
Software Security, Inc.	141	222
Software Support	114	224
Softway	68	226
Solution Systems	58	228
Solution Systems	113	229
Stepstone	131	231
Summit Information Systems, Inc.	65	233
Suncloud Software, Inc.	77	235
Tenon Software	73	*
Tool Makers (The)	81	239
Turbo Tech Report	32A	*
Unipress	51	243
V Communications	63	245
VegaCon Corporation	69	247
Vermont Creative Software	40	*
Vesta Technology Inc.	77	249
Visible Systems	9	251
Watcom C for IBM PCs	57	253
Wyte Corporation	71	255
Zortech, Inc.	55	257

*The advertiser prefers to be contacted by phone; consult ad.

Advertising Sales Offices

Midwest Charles Shively (415) 366-3600
Northeast Cynthia Zuck (914) 762-9552
Martha Brandt (415) 366-3600
Northern California/Northwest Lisa Boudreau (415) 366-3600
Southern California/AZ/NM/TX Michael Wiener (415) 366-3600
Telemarketing Rep./SE/SW USA Cheri Blum (714) 761-0294
Director of Marketing and Advertising Ferris Ferdon (415) 366-3600

OF INTEREST

(continued from page 136)

(REGA) plug-in model is available.

The RGDI Developer's Kit is available for $695. The optional Advanced RGDI Developer's Kit add-on is priced at $495. The REGA option costs $169. Reader Service No. 25.

Renaissance GRX
Cedar Park
2265 116th Ave. NE
Bellevue, WA 98004
206-454-8086

SoftScience Corp. has released its Convenience Plus DOS Front End which is designed for use with IBM's new 3363 Optical Disk. The program is intended for the novice and advanced PC user as a front end to MS-DOS, PC-DOS, and OS/2 and is designed for supporting file management and recovery on the IBM 3363 Optical Disk.

The program features the ability to perform DOS commands and additional commands not available through DOS; the ability to organize and understand the arrangement graphically of the computer; the ability to use the computer without memorizing or typing complicated syntax and language at a faster pace; unusual file recovery and management commands for the IBM Optical Disk; and DOS commands in five foreign languages. Reader Service No. 26.

SoftScience Corp.
Box 42905
Tucson, AZ 85733-2905
602-326-4679

Sourcer, available from **V Communications**, allows programmers to create commented source code and listings directly from RAM, ROM, .COM files and .EXE files. Sourcer creates detailed commented listings and source code directly suitable for assembly. Built-in data analyzer and code simulator resolves data items across multiple data segments, provides detailed comments on BIOS and MS-DOS interrupt calls and subfunctions, and I/O ports. Sourcer also determines proper assembler directives for multi-segment programs. Built in processor filter optimizes code based on instruction set selected, 80286, 80186/88, 8088/86 and V20/V30.

The Sourcer is also available with the BIOS Pre-Processor, which provides the first means to obtain accurate legal source listings for any BIOS. It identifies entry points with detailed in-line comments explaining functions and subfunctions, registers, and other key information.

The Sourcer costs $99.95, the Sourcer with BIOS Pre-Processor costs $139.95. Reader Service No. 27.

V Communications
3031 Tisch Wy., Ste. 200
San Jose, CA 95128
408-296-4224

Blaise Computing has announced ASYNCH PLUS/4.0, a comprehensive set of routines designed for Turbo Pascal 4.0 to give programmers the power to create interrupt driven communications software. The program has a layered design of separately compiled units, with the higher levels building on the lower levels. These routines drive virtually any asynchronous device via the RS-232 ports. ASYNCH PLUS includes low level control and queue maintenance functions written in assembly language and high-level routines written in Turbo Pascal to help programmers develop communication software. Fully documented source code is included as well as a comprehensive indexed manual which gives a general overview for every function category and descriptions of each function. Examples in the manual and full programs on the distribution diskettes serve as illustrations. ASYNCH PLUS is priced at $129. Reader Service No. 28.

Blaise Computing
2560 Ninth St., Ste. 316
Berkeley, CA 94710
415-540-5441

Programs in Motion offers a shortcut to large-system expert system developers by parlaying its facile, direct handling of decision trees with its new ability to generate production rules in a variety of programming languages. This expert system development software can work as a scratch pad for quick decision-tree prototyping, or even as a working breadboard. Once a decision tree functions as desired, the software

Multi-Edit vs. PIZZA

With EVERYTHING!

- Is your editor OUT TO LUNCH?
- Does it handle ALL OF YOUR NEEDS?
- Is it flexible, programmable and reconfigurable?
- MOST IMPORTANTLY, is it EASY TO USE?

OR WOULD YOU RATHER BE EATING PIZZA?

MULTI-EDIT $99 COMPLETE

VERSION 2

	Multi-Edit	BRIEF 2.0	Norton Editor	Vedit Plus	PIZZA WITH EVERYTHING
Edit 20+ Files larger than memory	Yes	Yes	No	Yes	12 slices
Powerful high level macro language	Yes	Yes	No	Yes	Italian
Full UNDO	Yes	Yes	No	No	No
Visual marking of blocks	Yes	Yes	Yes	No	Looks Good
Line, stream and column blocks	Yes	Yes	No	No	Use Knife
Automatic file save	Yes	Yes	No	No	No
Online help	Extensive	Limited	Limited	Limited	Extensive
Choice of keystroke commands or menu system	Yes	No	No	Yes	Menu Available
Function Key assignments labeled on screen (may be disabled)	Yes	No	No	No	No
Word processing functions	Extensive	Limited	Limited	Extra Cost	Difficult
Complete DOS shell	Yes	No	No	No	Deep Dish
Pop-up Programmer's Calculator and ASCII Table	Yes	No	No	No ASCII	No
Unlimited 'Off the Cuff' keystroke macros	Yes	No	No	Yes	Sauce on Cuff often
Allocates all available memory to compiler when run from within editor	Yes	No	No	No	Lots of bytes
Intelligent indenting, template editing and brace/parenthesis/block matching and checking for C, PASCAL, BASIC and MODULA-2	Yes	C Only	No	Limited	Limited Intelligence
Flexible condensed mode display	Yes	No	Yes	No	Definitely
PRICE	$99	$195	$50	$185	About $12

Get Our FULLY FUNCTIONAL DEMO Copy for only $10!

To Order, Call 24 hours a day:
1-800-221-9280 Ext. 951
In Arizona: 1-602-968-1945
Credit Card and COD orders accepted.

American Cybernetics
1228 N. Stadem Dr.
Tempe, AZ 85281

Requires IBM/PC/XT/AT/PS2 or full compatible, 256K RAM, PC/MS-DOS 2.0 or later. Multi-Edit and American Cybernetics are trademarks of American Cybernetics. BRIEF is a trademark of Underware, Inc. Norton Editor is a trademark of Peter Norton Computing, Inc. Vedit is a registered trademark of CompuView Products Inc. Copyright 1987 by American Cybernetics.

Only MULTI-EDIT tastes this good!

Fully automatic Windowing and Virtual Memory
- Edit multiple files regardless of physical memory size
- Easy cut-and-past between files
- View different parts of the same file

Powerful, EASY-TO-READ high-level macro language
- Standard language syntax
- Full access to ALL Editor functions

Language-specific macros for C, PASCAL, BASIC and MODULA-2
- Smart Indenting
- Smart brace/parenthesis/block checking
- Template editing
- More languages on the way

Terrific word-processing features for all your documentation needs
- Intelligent word-wrap
- Automatic pagination
- Full print formatting with justification, bold type, underlining and centering
- Flexible line drawing
- Even a table of contents generator

Compile within the editor
- Automatically positions cursor at errors
- Allocates all available memory to compiler

Complete DOS Shell.
- Scrollable directory listing
- Copy, Delete and Load multiple files with one command
- Background file printing

Regular expression search and translate
Condensed Mode display, for easy viewing of your program structure
Pop-up FULL-FUNCTION Programmer's Calculator and ASCII chart

and MOST IMPORTANT, the BEST USER-INTERFACE ON THE MARKET!
- Extensive context-sensitive help
- Choice of full menu system or logical function key layout
- Function keys are always labeled on screen (no guessing required!)
- Keyboard may be easily reconfigured and re-labeled
- Extensive mouse support
- Easy, automatic recording and playback of keystrokes
- Anchovies easily removed

MULTI-EDIT COMBINES POWER WITH EASE OF USE LIKE NO OTHER EDITOR ON THE MARKET TODAY.

(continued from page 138)

can generate corresponding production rules of program code in C or Pascal. In most cases, these modules can transfer into big expert system environments with little or no change. 1st-Class Fusion for IBM PCs, IBM XT, IBM ATs, or compatibles is priced at $1,295. Reader Service No. 29.

Programs in Motion
286 Boston Post Rd.
Wayland, MA 01778
617-358-7722

The GSS Graphics Development Toolkit for OS/2 is available from **Graphic Software Systems**. The GSS Graphics Development Toolkit for OS/2 provides a high-performance graphics development environment for OS/2-based personal computers. Its high-level functions speed development of interactive graphics applications, and a growing set of device drivers removes the burden of writing driver code for input, display, and output devices. The Graphics Development Toolkit supports advanced features of OS/2 and maintains source code compatibility with the Graphic Development Toolkit for DOS.

The GSS Graphics Development Toolkits for DOS and OS/2 are priced at $495 and $695, respectively. Reader Service No. 30.

Graphic Software Systems
9590 SW Gemini Dr.
P.O. Box 4900
Beaverton, OR 97005
503-641-2200

Tools and Utilities

Flambeaux Software has announced TECH Help! Version 3.3A. TECH Help! is an on-screen reference for system-level programmers. It includes coverage of the DOS and ROM-BIOS services, system variables, I/O ports, installable device drivers, and the layouts and structures of dozens of data tables, bit flags, and switch settings. It covers some topics which are not documented in the official reference manuals. The new version covers DOS 3.3 and the latest versions of the ROM-BIOS.

The display driver has been upgraded to include user-configurable color selection, a simplified way to access multiple Help! manuals, and increased display speed for EGA and VGA monitors.

TECH Help! runs on computers that are compatible with the IBM PC, IBM XT, IBM AT, and IBM PS/2 computers. The program is priced at $89.95. Reader Service No. 31.

Flambeaux Software
1147 E. Broadway, Ste. 56
Glendale, CA 91205
818-500-0044

California 10 PAK, by **California Software Products**, has been upgraded to be used with OS/2 on the IBM PS/2 and compatible machines. California 10 PAK contains 16 programs for browsing, comparing, and sorting the contents of files and memory. System configuration and a map of all installed memory may be displayed. A disassembler produces ready-to-edit-and-assemble source files from .COM files, .EXE files, or any area of main memory. An operating system shell allows users to define the operation of function keys and to create color menus and help screens. California 10 PAK runs under any version of DOS and under OS/2 in protected or unprotected modes. The price for the product is $79. Reader Service No. 32.

California Software Products
525 N. Cabrillo Park Dr.
Santa Ana, CA 92701-5017
714-973-0440

DDJ

golden bow products

Vcache	60	**55**
Vfeature *Hard Disk Utility*	80	**74**
Vfeature Deluxe	120	**111**
Vopt *Hard Disk Optimization Utility*	60	**55**

greenleaf products

Greenleaf C Sampler *specify QuickC or Turbo C*	95	**69**
Greenleaf Comm Library	185	**125**
Greenleaf Data Windows *Includes Source Code*	295	**189**
for OS/2 ... *New*	395	**249**
Greenleaf Functions	185	**125**

komputerwerk products

Finally *BASIC Routines*	99	**85**
Finally Modules	99	**85**
Finally XGraf	99	**85**

lattice products

Lattice C Compiler *from Lattice* ... *New Version*	450	**239**
with Library Source Code	900	**495**
C Cross Reference Generator	50	**37**
C-Food Smorgasbord *Function Library*	150	**95**
with Source Code	300	**179**
C-Sprite *Source Level Debugger*	175	**119**
Curses *Screen Manager*	125	**85**
with Source Code	250	**169**
dBC III	250	**159**
with Source Code	500	**318**
dBC III Plus	750	**594**
with Source Code	1500	**1184**
LMK *Make Facility*	195	**138**
RPG II Combo *All four items below*	1400	**1099**
RPG II Compiler *No Royalties*	750	**625**
Screen Design Aid Utility *for RPG II*	350	**309**
SEU *Source Entry Utility*	250	**199**
Sort/Merge	250	**199**
Text Management Utilities	120	**88**

meridian products

AdaVantage *DOD-validated* ... *New Version*	795	**735**
with Optimizer	995	**919**
AdaVantage Debugger	495	**449**
AdaVantage DOS Environment Package	50	**47**
AdaVantage Utility Packages	50	**47**

Curses
by Aspen Scientific

Binary Version:
List $119 Ours $105
With Source Code:
List $289 Ours $249

Curses is a DOS version of the C tools distributed with UNIX System V, UNIX 4.2 BSD and XENIX. You can create, paint, scroll, pop up, pull down, tile and overlay screen text-oriented windows at a fraction of what it would cost to develop such tools inhouse. Includes F.A.S.T. Development Toolkit for WYSIWYG forms design, and a C runtime library.

metagraphics products

MetaWINDOW *No Royalties*	195	**138**
MetaWINDOW/PLUS	275	**195**
QuickWINDOW/C *for Microsoft QuickC*	95	**79**
TurboWINDOW/C *for Borland Turbo C*	95	**79**
TurboWINDOW/Pascal *for Borland Turbo Pascal*	95	**79**

metaware products

High C ... *New*	595	**CALL**
386 Version ... *New*	895	**CALL**
Professional Pascal ... *New*	595	**CALL**
386 Version ... *New*	895	**CALL**

microport products

DOSMerge286 *Specify 2-Users or Unlimited*	149	**129**
DOSMerge386 *2-Users*	395	**345**
DOSMerge386 *Unlimited Users*	495	**429**
System V/386 Combination	799	**669**
386 Runtime System	199	**169**
386 Software Development System	499	**429**
Text Preparation System	199	**169**
System V/AT Combination	549	**465**
AT Runtime System	199	**169**
AT Software Development System	249	**209**
Text Preparation System	199	**169**

microsoft products

Microsoft BASIC Compiler *for XENIX*	695	**439**
Microsoft BASIC Interpreter *for XENIX*	350	**219**
Microsoft C Compiler 5 *w/CodeView* ... *New Version*	450	**285**
Microsoft COBOL Compiler *with COBOL Tools*	700	**439**
for XENIX	995	**639**
Microsoft Excel	495	**319**
Microsoft FORTRAN *Optimizing Comp* ... *New Version*	450	**285**
Microsoft FORTRAN *for XENIX*	695	**439**
Microsoft MACH 20	495	**329**
Microsoft Macro Assembler ... *New Version*	150	**99**
Microsoft Mouse *with Paint & Mouse Menus*	150	**99**
with Microsoft Windows & Paint	200	**139**
with EasyCAD	175	**119**
Microsoft OS/2 Programmer's Toolkit ... *New*	350	**CALL**
Microsoft Pascal Compiler ... *New Version*	300	**189**
for XENIX	695	**439**
Microsoft QuickBASIC	99	**69**
Microsoft QuickC	99	**69**
Microsoft Windows	99	**69**
Microsoft Windows 386	195	**129**
Microsoft Windows Development Kit	500	**319**
Microsoft Word	450	**285**
Microsoft Works	195	**129**

mks products

MKS AWK	75	**65**
MKS RCS *Revision Control System*	189	**155**
MKS SQPS *SoftQuad Publishing Software* ... *New*	495	**379**
MKS Toolkit *with MKS VI Editor*	169	**129**
MKS Trilogy *with AWK, CRYPT & Korn Shell*	119	**99**
MKS VI *Editor*	75	**65**

PC/Forms
by Golden Solutions

List $150 Ours $129

PC/Forms is a high powered screen management package. PC/Forms takes the hassle out of screen design, screen management, and input data validation. Forms are created and maintained using the form editor; and processed at runtime via the PC/Forms runtime library. Using PC/Forms, the code required to process a complex screen can be reduced from several hundred lines to only a few.

mmc ad system products

C Programmer's Toolbox I	80	**69**
C Programmer's Toolbox II	80	**69**
C Programmer's Toolbox Combo	130	**115**

modula-2 language

LOGITECH Modula-2 Development System	249	**199**
Modula-2 Compiler Pack	99	**75**
Modula-2 Toolkit	169	**139**
LOGITECH Modula-2 Window Pkg	49	**39**
MODULA-2 *by Stony Brook*	195	**169**
with Utilities	345	**299**

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	**98**
with PLUS & PC Paintbrush	149	**119**
with PLUS & CAD Software	189	**153**
with PLUS & CAD & Paint	219	**179**
with PLUS & First Publisher	179	**139**
LOGIMOUSE C7 *with PLUS Package*	119	**98**
with PLUS & PC Paintbrush	149	**119**
with PLUS & CAD Software	189	**153**
with PLUS & CAD & Paint	219	**179**
with PLUS & First Publisher	179	**139**
LOGITECH HiREZ Mouse *for High-res Screens* ... *New*	149	**119**
LOGITECH Series 2 Mouse *for IBM PS/2* ... *New*	99	**79**
Microsoft Mouse *See Microsoft Section*		

novell development products

Btrieve *ISAM Mgr with No Royalties*	245	**184**
Xtrieve *Query Utility*	245	**184**
Report Option *for Xtrieve*	145	**99**
Btrieve/N *for Networks*	595	**454**
Xtrieve/N	595	**454**
Report Option/N *for Xtrieve/N*	345	**269**
XQL	795	**599**

oregon software products

Pascal-2	229	**179**
SourceTools *Make and Version Control* ... *New*	595	**429**
Network Version ... *New*	1495	**CALL**

other products

ACTOR *by The Whitewater Group*	495	**419**
ApBasic *by CompTech Software*	100	**79**
DocuMotion *by NWP-Intelligent Solutions*	160	**139**
Heap Expander *by The Tool Makers*	60	**55**
Interactive EASYFLOW *by Haventree*	150	**125**
JAM *by JYACC* ... *New*	750	**679**
MASTER*KEY *by Sharpe Systems*	80	**69**
Opt-Tech Sort *by Opt-Tech Data Proc*	149	**99**
pcAnywhere *by EKD Computer*	99	**89**
Source Print *by Aldebaran Labs*	97	**74**
Teamwork/PCSA *by Cadre Technologies*	995	**929**
TeleSwitch *by EKD Computer*	289	**229**
TLIB *Version Control System by Burton*	100	**89**
Tmark *by Tangent Designs*	80	**69**
Tree Diagrammer *by Aldebaran Labs*	77	**59**
TurboGeometry *by Disk Software*	100	**89**
TurboHALO *by IMSI, Specify Turbo C or Pascal*	80	**75**

peter norton products

Advanced Norton Utilities	150	**89**
Norton Commander	75	**55**
Norton Editor ... *New Version*	75	**55**
Norton Guides *Specify Language*	100	**65**
for OS/2 ... *New*	150	**109**
Norton Utilities	100	**59**

CIRCLE NO. 192 ON READER SERVICE CARD

phoenix products

Pasm86 *Macro Assembler version 2.0*	195	**108**
Pdisk *Hard Disk & Backup Utility*	145	**99**
Pfantasy Pac *Phoenix Combo*	995	**595**
Pfinish *Execution Profiler*	395	**209**
Pfix86plus *Symbolic Debugger*	395	**209**
PforCe *Specify C Compiler*	395	**209**
PforCe++ *Specify C Compiler and C++*	395	**209**
Plink86plus *Overlay Linker*	495	**275**
Pmaker *Make Utility*	125	**78**
Pmate *Macro Text Editor*	195	**108**
Pre-C *Lint Utility*	295	**154**

pmi products

EmsStorage	49	**45**
Graphix	149	**119**
Macro2	89	**79**
ModBase	89	**79**
Repertoire	89	**74**
Repertoire/Btrieve Toolkit	149	**119**

polytron products

PolyMake *UNIX-like Make Facility*	149	**129**
PolyShell	149	**125**
PolyXREF *Complete Cross Reference Utility*	219	**185**
PVCS Corporate *Version Control System*	395	**329**
PVCS Personal	149	**129**

pop-up reference guides

Peabody *by Copia Intl, Specify Language*	100	**89**
Resident Expert *by Santa Rita, Specify Lang*	CALL	**CALL**

WATCOM C6.0
by Waterloo Computer Systems

Special Introduction Price
List $295 Ours $249

WATCOM C6.0 is a complete, high-performance development system. Along with the ANSI C Optimizing Compiler and ANSI C Run-time library, WATCOM C6.0 includes a powerful windowed interactive debugger, the WATCOM general-purpose editor, a high-performance Linker, an Object Librarian, MAKE, an Object Code Disassembler, and a copy of WATCOM Express C which is ideal for development and debugging. Special compiler directives provide flexible run-time calling conventions to allow efficient use with a wide range of libraries and other lan-

sco products

XENIX System V for PS/2	CALL	**CALL**
XENIX System V 286	1295	**979**
Development System	595	**479**
Operating System	595	**479**
Text Processing Package	195	**144**
XENIX System V 386	1595	**1179**
Development System	795	**595**
Operating System	795	**595**
Text Processing Package	195	**144**

software garden products

Dan Bricklin's Demo II	195	**179**
Dan Bricklin's Demo Program	75	**57**
Dan Bricklin's Demo Tutorial	50	**45**

software connection products

dB2c FILES	199	**179**
dB2c TOOLKIT	299	**249**
dB2c WINDOWS	99	**89**

texas instruments

Arborist *Decision Tree Software*	595	**519**
PC Scheme *Lisp*	95	**77**
Personal Consultant Easy	495	**435**
Personal Consultant Images	495	**435**
Personal Consultant Online	995	**869**
Personal Consultant Plus	2950	**2589**
Personal Consultant Runtime	95	**84**

text editors

Brief & dBrief Combo *from Solution Systems*	275	**CALL**
Brief	195	**CALL**
dBrief *Customizes Brief for dBASE III*	95	**CALL**
Epsilon *Emacs-like editor by Lugaru*	195	**147**
ME Text Editor *by Magma Stsrems* ... *New*	89	**75**
with Source Code ... *New*	189	**159**
Vedit Plus *by CompuView*	185	**128**
For XENIX ... *New*	285	**229**

turbopower products

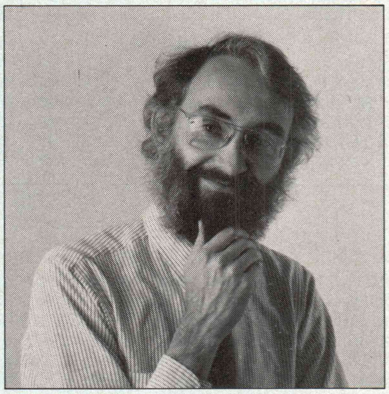
TDEBUG PLUS 4.0	45	**39**
with Source Code	90	**79**
Turbo Analyst 4.0	75	**59**
Turbo Professional 4.0	99	**79**

xenosoft products

XenoCopy-PC	80	69
XenoFont	50	45

CALL for Products Not Listed Here

SWAINE'S FLAMES



Everything is deeply intertwingled.
—Ted Nelson

I keep reading things that remind me of Ted Nelson's *ComputerLib/ Dream Machines* (either the original landmark manifesto à la collage of 1974 or the radically revised and recently rereleased Microsoft Press edition). There are several possible explanations.

First, Ted talks about a lot of things in *CL/DM*; his is an eclectic light.

Second, he mostly writes about things I mostly read about anyway: the potential of computers, the art of writing, the liberation of the mind.

Third, we remember unfinished business better than finished, and *CL/DM* is a business of unfinishedness; of loose ends; boxes that, once opened, can't be closed again; conceptual gambits; twitching, severed nerves; choose your metaphor. *CL/DM* pitches more itches than it hawks ointments for.

Fourth, everything is deeply intertwingled. That's probably it.

One thing that I've been reading that implicitly invokes *CL/DM* is *The Society of Mind* by Marvin Minsky. *The Society of Mind* presents a model of the mind as a society of communicating processes. The structure of the book reflects Minsky's model and reminds me of *CL/DM*. If you haven't read it, I recommend it.

I've also been reading about highly functionally distributed systems (HFDS): shades of Ted Nelson's *System Xanadu*. An HFDS is a heterogeneous loosely coupled worldwide network of computers and other "intelligent" objects, providing more sophisticated services than any of its components can, says University of Tokyo professor Ken Sakamura, who named HFDS. The multinational, multicompany project Sakamura spawned to implement it has a better name: TRON.

The TRON project envisions a global network and several types of networked devices, including intelligent objects and communication machines. The TRON team is designing a system for the day when a typical room contains a hundred computers, a building thousands, a city millions. The problems that would arise include questions like just how much should my neighbor's air conditioner know about my new lamp?

In an HFDS, neither centralized control nor anarchy would work. The cooperation of groups of components in an HFDS Nakamura intertwingledly calls the "society of computers."

The elements of the TRON project include ITRON, a spec for a real-time operating system for the control of intelligent objects; BTRON, an operating system spec for the human interface components of an HFDS; CTRON, a portable operating system spec for servers that will link BTRON and ITRON elements with gateways and large databases; and MTRON, which somehow ties it all together. The idea is that ITRON talks to machines, BTRON talks to people, CTRON talks to ITRON and BTRON machines, MTRON talks to Matsushita, and Matsushita talks only to God. I think.

Last November, TRON developers from more than 100 firms, including Matsushita, Fujitsu, AT&T, and IBM, met to report progress. Intended to go on-line in the 1990s, TRON is ahead of schedule.

I've been reading a lot about Apple's HyperCard lately, too, since I'm writing a book on it. The "hyper" is homage to Ted's hypertext, although he has some reservations re HyperCard as an implementation of same.

Others have also raised doubts about the market for HyperCard stackware. A panel at January's MacWorld show, discussing stackware prospects, was not encouraging. "Not until there are 100,000 CD-ROM units installed will there be a decent stackware market," one panelist concluded. Stewart Alsop thinks that unless developers, distributors, and Apple all treat stackware like full-fledged serious software, "The stackware business will disappoint just as many people as the 1-2-3 and dBASE templates business did." And Dvorak declares, "I wish those budding stackware developers a ton of luck. They'll need it."

I suspect that the world will little note nor long remember such sage skepticism, even if it turns out to be true. Since Apple brought forth on this continent the computer for the rest of us, Macintosh software development has been beyond the poor powers of unprofessional performers, but now any Sunday afternoon matinee walk-on bit player can prototype a product in a day or two, down to doing a cover letter in card form and printing the disk label.

Heaps of hyperstuff will be hacked together. Publishers will be pummeled in submissions if not into submission, and rejection will be no deterrent to the hyped-up hordes of hyperdom.

Reminds me of what Ted Nelson said about all that white-out on the screen. . . .

Michael Swaine

Michael Swaine

Employment Wanted

Fast worker!
No job too big!
References available:
Call (800) 541-1261

Ask for
Microsoft FORTRAN 4.1

Our FORTRAN is hungry for work. And it can handle any job you've got. Whether you're writing a math-intensive engineering program on your PC or porting a huge scientific application down from a mainframe, Microsoft® FORTRAN Optimizing Compiler version 4.1 is the FORTRAN for you.

It comes with an impressive resume and a great track record. And with new support for the OS/2 systems along with standard DOS support, it's a real team player that fits in with any organization.

Put your FORTRAN development on the fastest track yet.

Microsoft FORTRAN 4.1 generates the fastest FORTRAN programs on a personal computer. Our leading-edge optimizing technology, including loop optimizations, automatically improves the quality of the code it generates.

The result is object code so streamlined, so compact, and so efficient you may think your program is running on a mainframe.

Take on the biggest assignments with OS/2 systems support.

Support for the OS/2 systems gives you new capabilities. Write huge FORTRAN applications that shatter the 640K barrier of DOS. Make direct calls to the operating system. Even create a single "Family API" program that runs under MS® OS/2 *and* MS-DOS®.

When it comes to porting huge programs to and from mainframe and mini environments, Microsoft FORTRAN 4.1 can tackle projects of any size–even programs as large as one gigabyte!

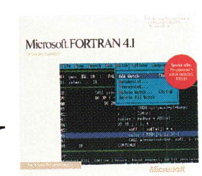
And it's still GSA-certified Full and error-free ANSI FORTRAN 77, with the largest set of IBM® VS and DEC® VAX® extensions available for personal computers.

So Microsoft FORTRAN 4.1 makes short work of any porting chore. And you're assured of the highest level of reliability you can get in a FORTRAN compiler.

You also get a great set of tools: the new Microsoft Editor, the famous Microsoft CodeView® debugger for both MS OS/2 and DOS, and award-winning documentation.

Give Microsoft FORTRAN 4.1 a long-term contract.

But first check its references for yourself. Just call **(800) 541-1261, Dept. E83,** for the name of your nearest Microsoft dealer.
Then put Microsoft FORTRAN 4.1 to work for you.



Microsoft FORTRAN
Optimizing Compiler version 4.1